

[illegible]

```

LL          IIIII
LL          IIIII
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LL          III
LLLLLLLLLLLL IIIII
LLLLLLLLLLLL IIIII
SSSSSSSSSS
SSSSSSSSSS
SS
SS
SS
SS
SSSSSS
SSSSSS
SS
SS
SS
SS
SSSSSSSSSS
SSSSSSSSSS

```

(2)	151	DECLARATIONS
(5)	293	EXE\$ENQ - Enqueue system service
(6)	424	CONVERSIONS
(7)	749	CVT_TO_SYS - Convert to system owned lock
(8)	826	CVT_TO_PRC - Convert to process owned lock
(9)	896	NEW_LOCK - New lock request (not conversion)
(12)	1532	Error Handling for \$ENQ
(13)	1681	LCK\$HASH_SEARCH - Hash resource and search hash table
(14)	1807	LCK\$GRANT_LOCK - Grant a lock request
(15)	2037	LOCK_KAST - Kernel AST routine
(16)	2186	LCK\$QUEUECVT - Insert a lock on conversion queue
(16)	2187	LCK\$QUEUEWAIT - Insert a lock on wait queue
(17)	2296	LCK\$QUEUE_BLOCKAST - Queue blocking ASTs
(18)	2439	LCK\$COMP_GMODE - Compute group grant mode
(19)	2501	LCK\$GRANTCVTS - Grant conversions
(19)	2502	LCK\$GRANTWTRS - Grant waiters
(20)	2607	VERIFYLOCKID - Verify lock id
(21)	2729	EXE\$DEQ - Dequeue system service
(22)	2964	LCK\$CANCEL_CVT - Cancel a waiting conversion
(23)	3040	LCK\$DEQLOCK - Dequeue a lock
(24)	3373	LCK\$CHECK_RSB - Deallocate RSB if necessary
(25)	3486	STALL_REQ - Stall request during failover
(26)	3551	LCK\$EXTEND_IDTBL - Extend lock id. table
(27)	3677	FREE_LKB - Free LKB (from AST queue)

```
0000 1 .TITLE SYSENQDEQ - ENQUEUE/DEQUEUE SYSTEM SERVICES
0000 2 .IDENT 'V04-000'
0000 3
0000 4 *****
0000 5 *
0000 6 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8 * ALL RIGHTS RESERVED.
0000 9 *
0000 10 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15 * TRANSFERRED.
0000 16 *
0000 17 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19 * CORPORATION.
0000 20 *
0000 21 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23 *
0000 24 *
0000 25 *****
0000 26
0000 27 **
0000 28 FACILITY: EXECUTIVE, SYSTEM SERVICES
0000 29
0000 30 ABSTRACT:
0000 31 This module implements the $ENQ, $ENQW, and $DEQ system services.
0000 32
0000 33 ENVIRONMENT: VAX/VMS
0000 34
0000 35 AUTHOR: Steve Beckhardt, CREATION DATE: 30-Oct-1980
0000 36
0000 37 MODIFIED BY:
0000 38
0000 39 V03-029 SRB0151 Steve Beckhardt 21-Aug-1984
0000 40 Fixed bug in 'parent not granted' code to avoid spurious
0000 41 errors.
0000 42
0000 43 V03-028 SRB0144 Steve Beckhardt 9-Aug-1984
0000 44 Fix broken branch.
0000 45
0000 46 V03-027 SRB0141 Steve Beckhardt 6-Aug-1984
0000 47 Changed the way system-owned locks are determined in
0000 48 VERIFYLOCKID to remove race condition that gave incorrect
0000 49 $$$_IVLOCKID errors.
0000 50
0000 51 V03-026 SRB0139 Steve Beckhardt 24-Jul-1984
0000 52 Added new entry point, LCK$CHECK_STALL to allow $GETLKI
0000 53 (and others) to stall during state transitions.
0000 54
0000 55 V03-025 SRB0137 Steve Beckhardt 11-Jul-1984
0000 56 Remove race conditions from dequeue/all code.
0000 57
```

0000	58	:	V03-024	SRB0133	Steve Beckhardt	22-Jun-1984
0000	59	:		Fixed two bugs: 1) Don't lose lock mode in conversion code		
0000	60	:		when an AST is already queued. 2) Check CVTSYS bit of parent		
0000	61	:		lock when converting a lock to system owned instead of		
0000	62	:		looking for a zero PID.		
0000	63	:				
0000	64	:	V03-023	SRB0132	Steve Beckhardt	25-May-1984
0000	65	:		Allowed parent locks to be in CONVERT state when \$ENQing		
0000	66	:		sublocks.		
0000	67	:				
0000	68	:	V03-022	SRB0126	Steve Beckhardt	9-May-1984
0000	69	:		Fixed bug whereby locks that were system owned that		
0000	70	:		were not successfully converted no longer had the		
0000	71	:		LCKSM_CVTSYS bit set.		
0000	72	:				
0000	73	:	V03-021	CWH3021	CW Hobbs	14-Apr-1984
0000	74	:		Fixed some broken branches.		
0000	75	:				
0000	76	:	V03-020	SRB0119	Steve Beckhardt	6-Apr-1984
0000	77	:		Changed access mode checking on lock ids. Reorganized		
0000	78	:		code involved in cancelling conversions and fixed bug.		
0000	79	:		Added support for LCKSM_NODLCKWT flag.		
0000	80	:				
0000	81	:	V03-019	SRB0116	Steve Beckhardt	8-Mar-1984
0000	82	:		Return status code in LCK\$DEQLOCK, change all SETIPL #0		
0000	83	:		to SETIPL #IPL\$ASTDEL, and fix two broken word displacements.		
0000	84	:				
0000	85	:	V03-018	LJK0264	Lawrence J. Kenz	29-Feb-1984
0000	86	:		Fix broken word displacements.		
0000	87	:				
0000	88	:	V03-017	SRB0108	Steve Beckhardt	11-Jan-1983
0000	89	:		Added support for hashed root directory.		
0000	90	:				
0000	91	:	V03-016	SRB0106	Steve Beckhardt	6-Dec-1983
0000	92	:		Changed LKBSL_REFCNT, RSB\$SL_REFCNT, RSB\$SL_BLKASTCNT		
0000	93	:		to word fields.		
0000	94	:				
0000	95	:	V03-015	SRB0101	Steve Beckhardt	7-Sep-1983
0000	96	:		Fixed bug that prevented canceling locks with sublocks.		
0000	97	:				
0000	98	:	V03-014	SRB0100	Steve Beckhardt	18-Jul-1983
0000	99	:		Enabled local deadlock detection. Fixed PMS counters.		
0000	100	:		Fixed code for canceling locks and use of routine FREE_LKB.		
0000	101	:				
0000	102	:	V03-013	SRB0094	Steve Beckhardt	23-Jun-1983
0000	103	:		Added support for PROTECT and RECOVER bits. Added support		
0000	104	:		for converting new locks to be system owned. Made several		
0000	105	:		changes to support multinode failover.		
0000	106	:				
0000	107	:	V03-012	SRB0090	Steve Beckhardt	20-May-1983
0000	108	:		Added support for extending lock id. table. Moved some		
0000	109	:		PMS counters and added new ones. Cleared CVTSYS bit instead		
0000	110	:		of ignoring it to fix bug involving system owned locks.		
0000	111	:		Added support for stalling requests during a failover.		
0000	112	:				
0000	113	:	V03-011	SRB0083	Steve Beckhardt	29-Apr-1983
0000	114	:		Added support for system owned locks. Rewrote portions		

```
0000 115 : of the conversion and dequeue code.
0000 116 :
0000 117 : V03-010 SRB0073 Steve Beckhardt 25-Mar-1983
0000 118 : Added support for two new $DEQ flags: CANCEL and INVVALBLK.
0000 119 :
0000 120 : V03-009 SRB0069 Steve Beckhardt 7-Mar-1983
0000 121 : Changed HALT to BUG_CHECK. Modified handling of value blocks
0000 122 : on conversions to return value block on conversions to same
0000 123 : lock mode. Changed access mode handling to deliver ASTs in
0000 124 : mode of caller. Added support for LCKSM_NOQUOTA flag.
0000 125 : Added conditionals around .PSECTS to allow loading for
0000 126 : debugging.
0000 127 :
0000 128 : V03-008 RNG0008 Rod N. Gamache 2-Feb-1983
0000 129 : Changed paged PSECT to Y$EXEPAGED PSECT.
0000 130 :
0000 131 : V03-007 SRB0061 Steve Beckhardt 7-Jan-1983
0000 132 : Added support for distributed lock conversions.
0000 133 :
0000 134 : V03-006 SRB0057 Steve Beckhardt 15-Dec-1982
0000 135 : Added support for distributed $ENQ of new locks,
0000 136 : distributed $DEQ, blocking ASTs, root directory
0000 137 : handling, etc.
0000 138 :
0000 139 : V03-005 SRB0055 Steve Beckhardt 6-Oct-1982
0000 140 : Fixed a number of small things that prevented service from
0000 141 : being loadable. Added coded to dequeue subtrees.
0000 142 :
0000 143 : V03-004 SRB0053 Steve Beckhardt 6-Oct-1982
0000 144 : Fixed bug causing improper handling of common event flags
0000 145 : on conversions.
0000 146 :
0000 147 : V03-003 KDM0002 Kathleen D. Morse 28-Jun-1982
0000 148 : Added $PRDEF and $$SDEF.
0000 149 :--
```

```
0000 151      .SBTTL  DECLARATIONS
0000 152 :
0000 153 : INCLUDE FILES:
0000 154 :
0000 155 :
0000 156 :
0000 157 : EXTERNAL SYMBOLS:
0000 158 :
0000 159 :
0000 160 :
0000 161 : MACROS:
0000 162 :
0000 163      $ACBDEF      ; ACB offsets
0000 164      $CADEF      ; Conditional assembly switches
0000 165      $DYNDEF     ; Structure type code definitions
0000 166      $IPLDEF     ; IPL definitions
0000 167      $IRPDEF     ; IRP offsets
0000 168      $JIBDEF     ; JIB offsets
0000 169      $LCKDEF     ; LCK definitions
0000 170      $LKBDEF     ; LKB offsets
0000 171      $PCBDEF     ; PCB offsets
0000 172      $PRDEF      ; Processor register definitions
0000 173      $PRIDEF     ; Priority increment class definitions
0000 174      $PRVDEF     ; Privilege bits
0000 175      $PSLDEF     ; PSL definitions
0000 176      $RSBDEF     ; RSB offsets
0000 177      $RSNDEF     ; Resource numbers
0000 178      $SSDEF      ; System status code definitions
0000 179 :
0000 180 : EQUATED SYMBOLS:
0000 181 :
0000 182 :
0000 183 :
0000 184 : Enqueue system service argument list offsets:
0000 185 :
0000 186 :
00000004 0000 187 EFN = 4      ; Event flag number
00000008 0000 188 LKMODE = 8   ; Lock mode
0000000C 0000 189 LKSB = 12    ; Lock status block address
00000010 0000 190 FLAGS = 16   ; Flags
00000014 0000 191 RESNAM = 20  ; Resource name
00000018 0000 192 PARID = 24   ; Parent id
0000001C 0000 193 ASTADR = 28  ; AST routine address
00000020 0000 194 ASTPRM = 32  ; AST routine parameter
00000024 0000 195 BLKAST = 36  ; Blocking AST address
00000028 0000 196 ACMODE = 40  ; Access mode
0000002C 0000 197 PROT = 44   ; Protection mask
0000 198 :
0000 199 :
0000 200 : Dequeue system service argument list offsets
0000 201 :
0000 202 :
00000004 0000 203 LOCKID = 4    ; Lock id
00000008 0000 204 VALBLK = 8   ; Value block address
0000000C 0000 205 DEQ_ACMODE = 12 ; Access mode
00000010 0000 206 DEQ_FLAGS = 16 ; Flags
0000 207 :
```

SYSENQDEQ
V04-000

- ENQUEUE/DEQUEUE SYSTEM SERVICES M 14
DECLARATIONS

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 5
(2)

```
0000000F 0000 208 POOL_MASK = ^XF ; Pool allocation granularity mask
          0000 209
          0000 210 :
          0000 211 : OWN STORAGE:
          0000 212 :
          0000 213
          0000 214 .IF NDF LOADSW
00000000 215 .PSECT LOCKMGR, LONG
          0000 216 .IFF
          0000 217 .PSECT $$$020
          0000 218 .ENDC
```


0000 220 :
0000 221 :
0000 222 :
0000 223 :
0000 224 :
0000 225 :
0000 226 :
0000 227 :
0000 228 :
0000 229 :
0000 230 :
0000 231 :
0000 232 :
0000 233 :
0000 234 :
0000 235 :
0000 236 :
0000 237 :
0000 238 :
0000 239 :
0000 240 :
0000 241 :
0000 242 :
0000 243 :
0000 244 :
0000 245 :
0000 246 :
0000 247 :
0000 248 :
0000 249 :
0000 250 :
0000 251 :
0000 252 :
0000 253 :
0000 254 :
3F 0000 255 :
1F 0001 256 :
07 0002 257 :
0B 0003 258 :
03 0004 259 :
01 0005 260 :

LOCK MODE COMPATIBILITY TABLE

The lock mode compatibility table represents the following compatibility matrix.

	NL	CR	CW	PR	PW	EX
	NL/SW	R/SW	W/SW	R/SR	W/SR	W/NL
NL NL/SW	yes	yes	yes	yes	yes	yes
CR R/SW	yes	yes	yes	yes	yes	no
CW W/SW	yes	yes	yes	no	no	no
PR R/SR	yes	yes	no	yes	no	no
PW W/SR	yes	yes	no	no	no	no
EX W/NL	yes	no	no	no	no	no

The compatibility table makes the following assumptions regarding the values of the lock modes. These values cannot be changed without changing the table

ASSUME LCK\$K_NLMODE EQ 0
ASSUME LCK\$K_CRMODE EQ 1
ASSUME LCK\$K_CWMODE EQ 2
ASSUME LCK\$K_PRMODE EQ 3
ASSUME LCK\$K_PWMODE EQ 4
ASSUME LCK\$K_EXMODE EQ 5

LCK\$COMPAT_TBL::

.BYTE ^B 111111
.BYTE ^B 011111
.BYTE ^B 000111
.BYTE ^B 001011
.BYTE ^B 000011
.BYTE ^B 000001

```

0006 262 :
0006 263 :
0006 264 :
0006 265 : This table indicates which conversions are always synchronous as
0006 266 : opposed to those which may be asynchronous.
0006 267 :
0006 268 :
0006 269 :
0006 270 :
0006 271 :
0006 272 :
0006 273 :
0006 274 :
0006 275 :
0006 276 :
0006 277 :
0006 278 :
0006 279 :
0006 280 :
0006 281 :
0006 282 :
0006 283 :
0006 284 :
0006 285 :
0006 286 :
0006 287 :
0006 288 :
0006 289 :
0006 290 :
0006 291 :
01 0006 286 :
03 0007 287 :
07 0008 288 :
0B 0009 289 :
1F 000A 290 :
3F 000B 291 :

```

```
000C 293 .SBTTL EXE$ENQ - Enqueue system service
000C 294
000C 295 :++
000C 296 : FUNCTIONAL DESCRIPTION:
000C 297 :
000C 298 : This routine handles the $ENQ system service.
000C 299 :
000C 300 : CALLING SEQUENCE:
000C 301 :
000C 302 : CALLS/G EXE$ENQ (actually called through the system service
000C 303 : dispatcher)
000C 304 :
000C 305 : INPUT PARAMETERS:
000C 306 :
000C 307 : EFN(AP) Event flag number
000C 308 : LKMODE(AP) Lock mode
000C 309 : LKSB(AP) Address of lock status block
000C 310 : FLAGS(AP) Flags
000C 311 : RECNAM(AP) Address of descriptor of resource name
000C 312 : PARIU(AP) Parent lock id
000C 313 : ASTADR(AP) Address of completion AST routine
000C 314 : ASTPRM(AP) AST parameter
000C 315 : BLKAST(AP) Address of blocking AST routine
000C 316 : PROT(AP) Protection mask
000C 317 : ACMODE(AP) Access mode
000C 318 :
000C 319 : R4 Address of PCB
000C 320 :
000C 321 : OUTPUT PARAMETERS:
000C 322 :
000C 323 : R0 Completion code
000C 324 :
000C 325 : COMPLETION CODES:
000C 326 :
000C 327 : In R0:
000C 328 :
000C 329 : SSS_NORMAL Successful completion
000C 330 : SSS_SYNCH Synchronous successful completion
000C 331 : SSS_ACCVIO Access violation (on LKSB or resource name)
000C 332 : SSS_BADPARAM Bad lock mode
000C 333 : SSS_IVLOCKID Invalid lock id
000C 334 : SSS_CVTUNGRANT Attempted to convert an ungranted lock
000C 335 : SSS_PARNOTGRANT Parent lock not granted
000C 336 : SSS_NOSYSLCK No SYSLCK privilege (needed for a system lock)
000C 337 : SSS_IVBUFLN Resource name length = 0 or > 31
000C 338 : SSS_INSFMEM Insufficient dynamic memory
000C 339 : SSS_EXASTLM Exceeded AST quota
000C 340 : SSS_EXENQLM Exceeded enqueue quota
000C 341 : SSS_NOTQUEUED Request was not queued
000C 342 : SSS_EXDEPTH Exceeded allowed depth of resource name tree
000C 343 : SSS_NOPRIV No privilege (to not charge quota or convert to
000C 344 : system owned lock)
000C 345 : SSS_SUBLOCKS Attempted to convert a system owned lock with
000C 346 : sublocks
000C 347 : SSS_PARNOTSYS Parent lock not system owned
000C 348 :
000C 349 : In LKSB:
```

```
000C 350 :  
000C 351 : SSS_NORMAL Successful completion  
000C 352 : SSS_DEADLOCK Deadlock detected  
000C 353 : SSS_ABORT Lock was dequeued before being granted  
000C 354 : SSS_CANCEL Lock request was canceled before being granted  
000C 355 :  
000C 356 :--  
000C 357 :  
000C 358 :.IF NDF LOADSW  
0000 0000 359 :.PSECT Y$EXEPAGED  
0000 360 :.ENDC  
0000 361 :  
0000 362 :.ENABL LSB  
0000 363 :  
00000000'GF 16 0000 364 5$: JSB G^SCH$GETEFC ; Validate common event flag  
53 04 AC D0 0006 365 : MOVL EFN(AP),R3 ; Refetch event flag number  
OC 50 E8 000A 366 : BLBS R0,10$ ; Rejoin common code  
04 000D 367 : RET ; Return - bad event flag  
000E 368 :  
000E 369 :.IF NDF LOADSW  
OFFC 000E 370 :.ENTRY EXE$ENQ,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>  
0010 371 :.IFF  
0010 372 :.ENTRY EXE$ENQ,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>  
0010 373 :.ENDC  
0010 374 :  
0010 375 : ; Verify event flag is ok  
0010 376 :  
53 04 AC D0 0010 377 : MOVL EFN(AP),R3 ; Get event flag number  
3F 53 91 0014 378 : CMPB R3,#63 ; Is it a local event flag?  
E7 1A 0017 379 : BGTRU 5$ ; No - validate common event flag  
0019 380 :  
0019 381 10$: ; Probe lock status block. For performance reasons, we assume the  
0019 382 : ; presence of a value block and probe 24 bytes. If this fails,  
0019 383 : ; then we check to see if there isn't a value block and in  
0019 384 : ; that case probe 8 bytes.  
0019 385 :  
0019 386 : ASSUME FLAGS EQ LKSB+4  
0019 387 : ASSUME LCK$V_VALBLK EQ 0  
0019 388 :  
58 0C AC 7D 0019 389 : MOVQ LKSB(AP),R8 ; Get address of lock status block in  
001D 390 : ; R8 and get flags in R9  
001D 391 : IFWRT #24,(R8),20$ ; Branch if writable  
1A 59 E8 0023 392 : BLBS R9,60$ ; Error if there's a value block  
0026 393 : IFNOWRT #8,(R8),60$ ; No value block - br. if not writable  
002C 394 :  
002C 395 20$: ; Get lock mode and check for legality  
002C 396 :  
57 08 AC 9A 002C 397 : MOVZBL LKMODE(AP),R7 ; Get lock mode  
05 57 91 0030 398 : CMPB R7,#LCK$K_EXMODE ; Is it legal?  
OE 1A 0033 399 : BGTRU 70$ ; No - error  
0035 400 :  
0035 401 : ; Determine if this is a new lock request or a conversion  
0035 402 :  
59 02 B3 0035 403 : BITW #LCK$M_CONVERT,R9 ; Is convert bit set?  
0038 404 :  
0038 405 :  
22 13 0038 406 : .IF NDF LOADSW  
BEQL NEW_LOCK ; Branch if new lock
```

```
00000049'GF 17 003A 407 JMP G^CONVERSION ; Conversion
                0040 408 .IFF
                0040 409 BNEQ CONVERSION ; Branch if conversion
                0040 410 BRW NEW_LOCK ; New lock
                0040 411 .ENDC
                0040 412
                0040 413 ;
                0040 414 ; ERRORS:
                0040 415 ;
                0040 416
0009 31 0040 417 60$: BRW ACCVIO ; Couldn't access LKSB
                0043 418
50 14 3C 0043 419 70$: MOVZWL S^#SS$ BADPARAM,R0 ; Bad lock mode
00000543'EF 17 0046 420 JMP ERROR_EXIT_R0
                004C 421
                004C 422 .DSAB' LSB
```

```

004C 424      .SBTTL  CONVERSIONS
004C 425
004C 426      :++
004C 427      : FUNCTIONAL DESCRIPTION:
004C 428
004C 429      :     This routine handles lock mode conversions.
004C 430
004C 431      : CALLING SEQUENCE:
004C 432
004C 433      :     Branched to from $ENQ service
004C 434      :     RET back to caller
004C 435
004C 436      : INPUTS:
004C 437
004C 438      :     R3      Event flag number
004C 439      :     R4      Address of PCB
004C 440      :     R7      Lock mode
004C 441      :     R8      Address of LKSB
004C 442      :     R9      Flags
004C 443
004C 444      :     Caller's argument list (offsets from AP)
004C 445
004C 446      : OUTPUTS:
004C 447
004C 448      :     R0      Completion code
004C 449
004C 450      : IMPLICIT OUTPUTS:
004C 451
004C 452      :     Caller's lock status block gets final request status (perhaps
004C 453      :     asynchronously)
004C 454
004C 455      : COMPLETION CODES:
004C 456
004C 457      :     In R0:
004C 458
004C 459      :     $$$_NORMAL      Successful completion
004C 460      :     $$$_SYNCH      Synchronous successful completion
004C 461      :     $$$_IVLOCKID     Invalid lock id
004C 462      :     $$$_CVTUNGRANT   Attempted to convert an ungranted lock
004C 463      :     $$$_INSFMEM      Insufficient dynamic memory
004C 464      :     $$$_EXASTLM      Exceeded AST quota
004C 465      :     $$$_NOTQUEUED     Request was not queued
004C 466      :     $$$_NOPRIV       No privilege (to convert to system owned lock)
004C 467      :     $$$_SUBLOCKS     Attempted to convert a system owned lock with
004C 468      :                     sublocks
004C 469      :     $$$_PARNOTSYS     Parent lock not system owned
004C 470
004C 471      :     In LKSB:
004C 472
004C 473      :     $$$_NORMAL      Successful completion
004C 474      :     $$$_DEADLOCK     Deadlock detected
004C 475      :     $$$_ABORT        Lock was dequeued before being granted
004C 476      :     $$$_CANCEL       Lock request was canceled before being granted
004C 477      : --
004C 478
004C 479      : .IF NDF LOADSW
0000000C 480      : .PSECT  LOCKMGR

```

```
000C 481 .ENDC
000C 482
000C 483 .ENABL LSB
000C 484
000C 485
000C 486 ; Errors:
000C 487
000C 488
50 0E32 8F 3C 000C 489 8$: MOVZWL #SS$_RETRY,R0 ; Retry operation
000C 490 BRB 12$
50 213C 8F 3C 0013 491 10$: MOVZWL #SS$_CVTUNGRANT,R0 ; Lock not currently granted
000C 492 12$: BRW ERROR_EXIT,R0
000C 493
000C 494 14$: BBS #LKBSV_DCPLAST,R7,16$ ; Branch if queued for completion AST
000C 495 PUSHL R5 ; Save lock mode
000C 496 MOVL R6,R5 ; Queued for blocking AST only
000C 497 JSB G$SCH$REMOVACB ; Remove it
000C 498 POPL R5 ; Restore lock mode
000C 499 BRB 21$ ; Continue with conversion
000C 500 16$: SETIPL #IPL$_ASTDEL ; Lower IPL
000C 501 JSB FREE_LKB ; Free up LKB
000C 502 BRB 15$ ; Returns at IPL$_SYNCH
000C 503
000C 504 17$: ; Stalling some requests - see which ones
000C 505
000C 506 BLSS 18$ ; Stalling all requests
000C 507 BBC #LKBSV_PROTECT,- ; Stalling only protected locks
000C 508 LKBSW STATUS(R6),19$ ; Branch if this is not a protected lock
000C 509 BBS #PCBSV_RECOVER,- ; Don't stall recovery process,
000C 510 PCBSL_STS(R4),8$ ; return error instead
000C 511 18$: BRW STALL_REQ ; Stall this request
000C 512
000C 513 CONVERSION:
000C 514 ; First get input arguments into registers (and on stack) because
000C 515 ; when they're stored we'll be at IPL$_SYNCH.
000C 516
000C 517 ASSUME ASTPRM EQ ASTADR+4
000C 518 ASSUME LCKSV_VALBLK EQ 0
000C 519
000C 520 MOVL R7,R5 ; Move lock mode to R5
000C 521 MOVQ ASTADR(AP),R10 ; Fetch completion AST address (R10)
000C 522 ; and AST parameter (R11)
000C 523 MOVL BLKAST(AP),R2 ; Fetch blocking AST address (R2)
000C 524 BLBC R9,15$ ; Branch if no value block
000C 525 MOVQ 16(R8),-(SP) ; Copy caller's value block onto stack
000C 526 MOVQ 8(R8),-(SP)
000C 527
000C 528 15$: ; Get lock id, validate and convert to LKB address in R6.
000C 529 ; Note that our raising IPL to IPL$_SYNCH is tied to the assumptions
000C 530 ; stated in the routine FREE_LKB.
000C 531
000C 532 MOVL 4(R8),R1 ; Fetch lock id
000C 533 SETIPL #IPL$_SYNCH ; Raise IPL
000C 534 BSBW VERIFYLOCKID ; Verify lock id and return LKB in R6,
000C 535 ; caller's access mode in R1
000C 536 BLBC R0,12$ ; Error
000C 537
```

```
00000000'GF 95 006C 538 ; Verify that we aren't stalling lock requests.
C6 12 006C 539
0072 540 TSTB G^LCK$GB_STALLREQS ; Are we stalling requests?
0074 541 BNEQ 17$ ; Yes, see which ones
0074 542 19$: ; Verify lock is granted and that the LKB is not queued to deliver
0074 543 ; an AST. If ok, store input args in LKB.
0074 544
0074 545 ASSUME LKBSW_STATUS EQ LKBSW_FLAGS+2
0074 546 ASSUME LKBSK_GRANTED GT 0
0074 547 ASSUME LKBSL_BLKASTADR EQ LKBSL_CPLASTADR+4
0074 548
0074 549
36 A6 95 0074 550 TSTB LKBSB_STATE(R6) ; Is the lock currently granted?
9A 15 0077 551 BLEQ 10$ ; No - error
57 2A A6 3C 0079 552 MOVZWL LKBSW_STATUS(R6),R7 ; Pick up current status
57 03 B3 007D 553 BITW #LKBSM_DCPLAST- ; Is the ACB portion of the LKB in use?
99 12 0080 554 !LKBSM_DBLKAST,R7 ; (are we delivering a completion or
AA 0082 555 BNEQ 14$ ; blocking AST?). Branch if yes.
0083 556 BICW #LKBSM_DCPLAST- ; Clear relevant status bits
0083 557 !LKBSM_DBLKAST- ; The following bits retain their
0083 558 !LKBSM_ASYNC- ; old state:
0083 559 !LKBSM_BLKASTQED- ; MSTCPY (shouldn't be set)
0083 560 !LKBSM_TIMEOUTQ- ; NOQUOTA
0083 561 !LKBSM_WASSYSOWN- ; PROTECT
0083 562 !LKBSM_CVTOSYS,-
0083 563 LKBSW_STATUS(R6)
2A A6 01CF 8F 90 0088 564 MOVB R3,LKBSB_EFN(R6) ; Store event flag number
37 A6 53 D0 008C 565 MOVL R8,LKBSL_LKSB(R6) ; Store LKSB address
24 A6 58 D0 0090 566 MOVL LKBSL_RSB(R6),R8 ; Get RSB address in R8
58 50 A6
0094 567
0094 568 ; Determine if we need to convert this lock to system owned or
0094 569 ; process owned. R1 contains caller's access mode.
0094 570
53 35 A6 9A 0094 571 MOVZBL LKBSB_GRMODE(R6),R3 ; Save old granted mode
0C A6 D5 0098 572 TSTL LKBSL_PID(R6) ; Is lock currently system owned?
13 13 009B 573 BEQL 22$ ; Yes
21 59 06 E1 009D 574 BBC #LCK$V_CVTSYS,R9,25$ ; No, handle normally if CVTSYS is clear
16 FF5F CF43 55 E1 00A1 575 BBC R5,LCK$SYNCCVT_TBL[R3],24$ ; Clear CVTSYS flag if async. cvt.
0120 30 00A8 576 BSBW CVT TO SYS ; Convert to system owned lock
59 08 A8 00AB 577 26$: BISW #LCK$M_SYNCSTS,R9 ; CVTSYS implies SYNCSTS so set it
12 11 00AE 578 BRB 25$
04 FF50 CF43 55 E1 00B0 579 22$: BBC R5,LCK$SYNCCVT_TBL[R3],23$ ; Cvt to process owned if async. cvt
F0 59 06 E0 00B7 580 BBS #LCK$V_CVTSYS,R9,26$ ; Leave as is if CVTSYS flag is set
014A 30 00BB 581 23$: BSBW CVT TO-PRC ; Convert to process owned lock
00 59 06 E4 00BE 582 24$: BBSC #LCK$V_CVTSYS,R9,25$ ; Clear CVTSYS flag, ignore branch
00C2 583
00C2 584 25$: ; All error checking and conversion between system owned and process
00C2 585 ; owned has been performed. Store new AST addresses and parameter.
00C2 586
20 A6 D0 00C2 587 MOVL LKBSL_BLKASTADR(R6),- ; Save old blocking AST address
5C A6 00C5 588 LKBSL_OLDBLKAST(R6)
00C7 589
00C7 590 ; Note: This tests the old contents
00C7 591 ; of this field and must come before
00C7 592 ; the new contents are stored.
14 A6 13 00C7 592 BEQL 30$ ; No
58 A6 D0 00C9 593 MOVL LKBSL_ASTPRM(R6),- ; Save old AST parameter
00CC 594 LKBSL_OLDASTPRM(R6)
```



```
14 42 A8 B7 00CE 595 30$: DECW RSB$W_BLKASTCNT(R8) ; Decr. blocking AST count
    6 5B D0 00D1 596      MOVL R11,LKB$S_ASTPRM(R6) ; Store new AST parameter
    5B 52 D0 00D5 597      MOVL R2,R11 ; Move blocking AST address
1C A6 5A 7D 00D8 598      MOVQ R10,LKB$S_CPLASTADR(R6) ; Store new completion AST address (R10)
    00DC 599 ; and new blocking AST address (R11)
28 A6 59 B0 00DC 600      MOVW R9,LKB$W_FLAGS(R6) ; Store flags
    51 55 D0 00E0 601      MOVL R5,R1 ; Move requested lock mode
    5B 53 D0 00E3 602      MOVL R3,R11 ; Move granted lock mode
    00E6 603 ;
    00E6 604 ; If a value block is specified and we are converted down (or same)
    00E6 605 ; from PW o 'X, then store caller's value block in RSB.
    00E6 606
    1A 59 E9 00E6 607      BLBC R9,33$ ; Branch if no value block specified
    04 5B 91 00E9 608      CMPB R11,#LCK$K_PWMODE ; Is granted mode PW or higher?
    15 1F 00EC 609      BLSSU 33$ ; No
    5B 51 91 00EE 610      CMPB R1,R11 ; Is conversion to a higher lock mode?
    10 1A 00F1 611      BGTRU 33$ ; Yes
28 A8 6E 7D 00F3 612      MOVQ (SP),RSB$Q_VALBLK(R8) ; No, copy caller's value block to RSB
30 A8 08 AE 7D 00F7 613      MOVQ 8(SP),RSB$Q_VALBLK+8(R8)
    3C A8 D6 00FC 614      INCL RSB$S_VALSEQNUM(R8) ; Increment value block sequence number
    02 AA 00FF 615      BICW #RSB$M_VALINVL,- ; Validate value block
    0E A8 0101 616      RSB$W_STATUS(R8)
    0103 617
    0103 618 33$: ; If this is a process copy LKB then jump to distributed lock
    0103 619 ; code.
    0103 620
    53 38 A8 D0 0103 621      MOVL RSB$S_CSID(R8),R3 ; Get CSID of destination system
    28 12 0107 622      BNEQ 35$ ; Yes
    0109 623
    00000002 0109 624      .IF NE CAS MEASURE
00000000'GF D6 0109 625      INCL G^PMS$GL_ENQCVT_LOC
    010F 626      .ENDC
    010F 627
    010F 628 LCK$LOCAL_CVT::
    010F 629 ; Remove this lock from the granted queue. If it was the only one and
    010F 630 ; if the conversion queue is also empty, then the conversion request
    010F 631 ; can be granted immediately. This path is special cased because it
    010F 632 ; is the normal case.
    010F 633
    50 38 A6 0F 010F 634      REMQUE LKB$S_SQFL(R6),R0 ; Remove lock from granted queue
    22 12 0113 635      BNEQ 40$ ; Not the only one
    5A 18 A8 DE 0115 636      MOVAL RSB$S_CVTQFL(R8),R10 ; It's the only granted lock
    5A 6A D1 0119 637      CMPL (R10),R10 ; Is conversion queue empty?
    19 12 011C 638      BNEQ 40$ ; It's not - must check the longer way
    04EF 30 011E 639      BSBW LCK$GRANT_LOCK_ALT ; It is - grant lock
    5A 08 C0 0121 640      ADDL #8,R10 ; Point to wait queue
    5A 6A D1 0124 641      CMPL (R10),R10 ; Is wait queue empty?
    42 13 0127 642      BEQL 60$ ; Yes, exit with completion status in R0
    5A 50 D0 0129 643      MOVL R0,R10 ; No, save status in R10
    07EC 30 012C 644      BSBW LCK$GRANTWTRS ; Try granting waiting locks
    37 11 012F 645      BRB 50$ ; Exit with completion status in R10
    0131 646
00000000'GF 17 0131 647 35$: JMP G^LCK$SND_CVTREQ ; Send convert request
    0137 648
    0137 649 ; Possible return points are:
    0137 650 ;
    0137 651 ; LCK$CVT_GRANTED ; Conversion was granted
```

```
0137 652 : LCK$QUEUE_EXIT Conversion was queued
0137 653 : LCK$CVTNOTQED Conversion was not queued
0137 654 : LCK$LOCAL_CVT Remote system failed and conversion
0137 655 : should now be done locally
0137 656
0137 657 40$: ; There was at least one other holder of the resource so we have
0137 658 ; to check for compatibility the longer way. The granted mode
0137 659 ; of this lock is compared with the conversion grant mode. If,
0137 660 ; other than the head of the conversion queue, there are granted
0137 661 ; locks with higher lock modes than this lock, then there
0137 662 ; is no need to recompute the group grant mode or attempt to
0137 663 ; grant waiting conversions.
0137 664
0D A8 5B 91 0137 665 CMPB R11,RSB$B_CGMODE(R8) ; Is granted mode = conv. grant mode?
10 13 013B 666 BEQL 45$ ; Yes
55 0C A8 9A 013D 667 MOVZBL RSB$B_GGMODE(R8),R5 ; No, get group grant mode
39 FEB9 CF45 51 E1 0141 668 BBC R1,LCK$COMPAT_TBL[R5],80$ ; Branch if not compatible
04C0 30 0148 669 BSBW LCK$GRANT_LOCK ; Grant the lock
1E 11 014B 670 BRB 60$ ; Exit with completion status in R0
014D 671
2A FEAA CF45 075F 30 014D 672 45$: BSBW LCK$COMP_GGMODE ; Compute new group grant mode in R5
51 E1 0150 673 BBC R1,LCK$COMPAT_TBL[R5],80$ ; Branch if not compatible
OC A8 55 90 0157 674 MOVB R5,RSB$B_GGMODE(R8) ; Store group grant mode in RSB
OD A8 55 90 015B 675 MOVB R5,RSB$B_CGMODE(R8) ; Also store conversion grant mode
04A9 30 015F 676 BSBW LCK$GRANT_LOCK ; Grant lock
5A 50 D0 0162 677 MOVL R0,R10 ; Save completion status in R10
076A 30 0165 678 BSBW LCK$GRANTCVTS ; Try granting conversions and waiters
50 5A D0 0168 679 50$: MOVL R10,R0 ; Restore completion status to R0
016B 680
016B 681 60$: ; The conversion was performed synchronously. Completion
016B 682 ; status is in R0. If a value block is specified then
016B 683 ; if we converted up or to the same level at NL - PR then
016B 684 ; return value block to caller.
016B 685
016B 686 ASSUME LCK$V_VALBLK EQ 0
016B 687
016B 688 LCK$CVT_GRANTED::
5B 10 59 E9 016B 689 BLBC R9,75$ ; Branch if no value block specified
35 A6 91 016E 690 CMPB LKB$B_GMODE(R6),P11 ; Was conversion to a lower lock mode?
0A 1F 0172 691 BLSSU 75$ ; Yes, don't return value block
05 1A 0174 692 BGTRU 65$ ; Cvt'd to a higher mode; return valblk
04 5B 91 0176 693 CMPB R11,#LCK$K_PWMODE ; Was old mode PW or higher?
03 1E 0179 694 BGEQU 75$ ; Yes, don't return value block
035A 31 017B 695 65$: BRW LCK$RET_VALBLK ; No, return value block to caller
037A 31 017E 696 75$: BRW LCK$NORET_VALBLK
0181 697
0181 698 80$: ; The conversion cannot be granted. Queue the request
0181 699 ; unless the noqueue bit is set. If the conversion queue is empty
0181 700 ; then R5 contains the new conversion grant mode.
0181 701
0A 59 02 E0 0181 702 BBS #LCK$V_NOQUEUE,R9,85$ ; Branch if noqueue is set
0185 703
0185 704 ; Queue the conversion
0185 705
34 A6 51 90 0185 706 MOVB R1,LKB$B_RMODE(R6) ; Store requested mode
0622 30 0189 707 BSBW LCK$QUEUECVT ; Insert onto conversion queue, etc.
0369 31 018C 708 BRW LCK$QUEUED_EXIT
```

```
018F 709
018F 710 85$: ; The request is not to be queued. Insert back onto the
018F 711 ; granted queue. R7 contains old LKBSW_STATUS.
018F 712
38 A6 OE 018F 713 INSQUE LKBSL_SQFL(R6),- ; Put lock back on granted queue
10 A8 0192 714 RSB$S_GLQFL(R8)
0194 715 LCK$CVTNOTQED::
0194 716 ; Restore old blocking AST address and parameter and requeue
0194 717 ; a blocking AST, if necessary.
0194 718
00000002 0194 719 .IF NE CAS MEASURE
00000000'GF D6 0194 720 INCL G^PMS$GL_ENQNOTQD
019A 721 .ENDC
019A 722
5C A6 D0 019A 723 MOVL LKBSL_OLDBLKAST(R6),- ; Restore old blocking AST address
20 A6 019D 724 LKBSL_BLKASTADR(R6)
14 13 C19F 725 BEQL 95$ ; None specified
58 A6 D0 01A1 726 MOVL LKBSL_OLDCASTPRM(R6),- ; Restore old AST parameter
14 A6 01A4 727 LKBSL_ASTPRM(R6)
42 A8 B6 01A6 728 INCW RSB$W_BLKASTCNT(R8) ; Incr. blocking AST count
08 57 01 E1 01A9 729 BBC #LKBSV_DBLKAST,R7,95$ ; Branch if blocking AST wasn't queued
55 56 D0 01AD 730 MOVL R6,R5
06CA 30 01B0 731 BSBW QUEUE_BLKAST ; Requeue blocking AST
OE 11 01B3 732 BRB 98$ ; Couldn't be system owned if a blocking
01B5 733 ; AST was queued.
01B5 734
01B5 735 95$: ; Now convert lock back to system owned, if necessary
01B5 736
07 E1 01B5 737 BBC #LKBSV_WASSYSOWN,- ; Branch if it wasn't system owned
09 2A A6 01B7 738 LKBSW_STATUS(R6),98$
54 00000000'GF D0 01BA 739 MOVL G^SCH$GL_CURPCB,R4 ; Get PCB address
18 10 01C1 740 BSBB CVT_TO_SYS_INT ; Convert to system lock
01C3 741
01C3 742 98$: ; Complete request with error status
01C3 743
50 09B8 8F 3C 01C3 744 MOVZWL #SS$ NOTQUEUED,R0 ; Store status
0378 31 01C8 745 BRW ERROR_EXIT_R0 ; Exit
01CB 746
01CB 747 .DSABL LSB
```

```
01CB 749 .SBTTL CVT_TO_SYS - Convert to system owned lock
01CB 750
01CB 751 :++
01CB 752 : FUNCTIONAL DESCRIPTION:
01CB 753 :
01CB 754 : This routine converts a lock from process owned to system owned
01CB 755 :
01CB 756 : CALLING SEQUENCE:
01CB 757 :
01CB 758 : BSBW CVT_TO_SYS
01CB 759 : BSBW CVT_TO_SYS_INT (Internal entry point w/o error checking)
01CB 760 : IPL must be at IPL$ SYNCH
01CB 761 : NOTE: Errors are passed to error exit, not returned to caller
01CB 762 :
01CB 763 : INPUTS:
01CB 764 :
01CB 765 : R1 Access mode of caller (CVT_TO_SYS entry only)
01CB 766 : R4 Address of PCB
01CB 767 : R6 Address of LKB
01CB 768 :
01CB 769 : OUTPUTS:
01CB 770 :
01CB 771 : R0 Completion code (returned to error handler, not caller)
01CB 772 :
01CB 773 : COMPLETION CODES:
01CB 774 :
01CB 775 : SSS_NOPRIV Caller wasn't in EXEC or KERNEL mode
01CB 776 : SSS_PARNOTSYS Parent lock is not a system lock
01CB 777 :
01CB 778 : SIDE EFFECTS:
01CB 779 :
01CB 780 : R0 is destroyed
01CB 781 :--
01CB 782 :
01CB 783 : .ENABL LSB
01CB 784 :
01CB 785 CVT_TO_SYS:
01CB 786 : Verify that caller is in EXEC or KERNEL mode and that
01CB 787 : this lock's parent lock is also system owned.
01CB 788 :
01 51 91 01CB 789 CMPB R1,#PSL$C_EXEC ; Is caller privileged?
50 48 A6 D0 01CE 790 BGTRU 70$ ; No, error
05 13 01D0 791 MOVL LKB$P_PARENT(R6),R0 ; Get parent LKB address
25 28 A0 E1 01D4 792 BEQL CVT_TO_SYS_INT ; No parent
01D6 793 BBC #LCK$V-CVTSYS,- ; Branch if parent not system owned
01D8 794 LKB$W_FLAGS(R0),80$
01DB 795
01DB 796 CVT_TO_SYS_INT:
01DB 797 : Request passed all error checks, do the conversion to system owned.
01DB 798 : Return quota to process, if charged, clear the lock's PID
01DB 799 : and remove from the process's lock queue.
01DB 800 :
01DB 801 BITW #LKB$M_NOQUOTA,- ; Was quota charged?
01DD 802 LKB$W_STATUS(R6)
01DF 803 BNEQ 20$ ; No
50 0080 C4 D0 01E1 804 MOVL PCB$JIB(R4),R0 ; Yes, get address of JIB
4C A0 B6 01E6 805 INCW JIB$W_ENQCNT(R0) ; Return quota
```

SYSENQDEQ
V04-000

M 15
- ENQUEUE/DEQUEUE SYSTEM SERVICES
CVT_TO_SYS - Convert to system owned loc

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 18
(7)

```

      20  A8 01E9 806      BISW  #LKBSM_NOQUOTA,-      ; Set NOQUOTA bit
      2A  A6 01EB 807      LKBSW_STATUS(R6)
      0C  A6 01ED 808
50    40  A6 D4 01ED 809 20$: CLRL  LKBSL_PID(R6)      ; Clear PID
      0040 8F 0F 01F0 810    REMQUE LKBSL_OWNQFL(R6),R0 ; Remove from PCB queue
      28  A6 A8 01F4 811    BISW  #LKBSM_CVTSYS,-      ; Set this flag in case we are
      01F8 812      LKBSW_FLAGS(R6)                  ; cancelling or we have a NOQUEUE
      01FA 813      ; situation (it got cleared)
      05 01FA 814      RSB
      01FB 815
      01FB 816
      01FB 817      ; Errors
      01FB 818
50    24  3C 01FB 819 70$: MOVZWL #SS$_NOPRIV,R0
      05  11 01FE 820      SRB  90$
50    225C 8F 3C 0200 821 80$: MOVZWL #SS$_PARNOTSYS,R0
      033B 31 0205 822 90$: BRW  ERROR_EXIT_R0
      0208 823
      0208 824      .DSABL LSB
```

```
0208 826      .SBTTL CVT_TO_PRC - Convert to process owned lock
0208 827
0208 828 :++
0208 829 : FUNCTIONAL DESCRIPTION:
0208 830 :
0208 831 :     This routine converts a lock from system owned to process owned
0208 832 :
0208 833 : CALLING SEQUENCE:
0208 834 :
0208 835 :     BSBW    CVT_TO_PRC
0208 836 :     IPL must be at IPL$_SYNCH
0208 837 :     NOTE:   Errors are passed to error exit, not returned to caller
0208 838 :
0208 839 : INPUTS:
0208 840 :
0208 841 :     R4      Address of PCB
0208 842 :     R6      Address of LKB
0208 843 :     R9      Input flags
0208 844 :
0208 845 : OUTPUTS:
0208 846 :
0208 847 :     R0      Completion code (returned to error handler, not caller)
0208 848 :
0208 849 : COMPLETION CODES:
0208 850 :
0208 851 :     SSS_EXENQLM    Exceeded enqueue quota
0208 852 :     SSS_SUBLOCKS   System owned locks with sublocks cannot be converted
0208 853 :                    to process owned locks
0208 854 :
0208 855 : SIDE EFFECTS:
0208 856 :
0208 857 :     R0 is destroyed
0208 858 :--
0208 859
0208 860 CVT_TO_PRC:
0208 861 : Verify that this lock has no sublocks
0208 862
0208 863 4C A6 B5 0208 863 TSTW    LKBSW_REFCNT(R6)      ; Are there any sublocks?
0208 864 2F 12 0208 864 BNEQ    80$                  ; Yes, error
0208 865 020D 865
0208 866 020D 866 ; Charge quota unless NOQUOTA bit is set
0208 867 020D 867
0208 868 59 20 B3 020D 868 BITW    #LKBSM_NOQUOTA,R9      ; Charge quota?
0208 869 0E 12 0210 869 BNEQ    20$                  ; No
0208 870 50 0080 C4 D0 0212 870 MOVL    PCB$L_JIB(R4),R0      ; Yes, get JIB address
0208 871 4C A0 B7 0217 871 DECV    JIBSW_ENQCNT(R0)      ; Charge one
0208 872 16 19 021A 872 BLSS    70$                  ; No quota - error
0208 873 20 AA 021C 873 BICW    #LKBSM_NOQUOTA,-      ; Indicate it was charged
0208 874 2A A6 021E 874 LKBSW_STATUS(R6)
0208 875 0220 875
0208 876 0220 876 20$: ; Store PID and insert onto PCB lock queue
0208 877 0220 877
0208 878 0080 8F A8 0220 878 BISW    #LKBSM_WASSYSOWN,-      ; Set status bit to indicate lock was
0208 879 2A A6 0224 879 LKBSW_STATUS(R6)      ; system owned
0208 880 60 A4 D0 0226 880 MOVL    PCB$L_PID(R4),-      ; Store PID
0208 881 0C A6 0229 881 LKBSL_PID(R6)
0208 882 40 A6 0E 022B 882 INSQUE  LKBSL_OWNOFL(R6),-      ; Insert at head of PCB queue
```

0104	C4	05	022E	883		PCBSL_LCCKQFL(R4)	
			0231	884	RSB		
			0232	885			
			0232	886	:		
			0232	887	:	Errors	
			0232	888	:		
			0232	889	:		
50	4C A0	B6	0232	890	70\$:	INCW	JIBSW_ENQCNT(R0)
	2A44 8F	3C	0235	891		MOVZWL	#SS\$_EXENQLM,R0
	05	11	023A	892		BRB	90\$
50	212C 8F	3C	023C	893	80\$:	MOVZWL	#SS\$_SUBLOCKS,R0
	02FF	31	0241	894	90\$:	BRW	ERROR_EXIT_R0

; Put back charged quota

```

0244 896 .SBTTL NEW_LOCK - New lock request (not conversion)
0244 897
0244 898 : FUNCTIONAL DESCRIPTION:
0244 899 :
0244 900 : This routine handles requests for new locks, as opposed to
0244 901 : conversions. This routine eventually branches to NEW_RESOURCE
0244 902 : or OLD_RESOURCE depending on whether the resource already
0244 903 : exists or not.
0244 904
0244 905 : CALLING SEQUENCE:
0244 906 :
0244 907 : Branched to from $ENQ service
0244 908 : RET back to caller
0244 909
0244 910 : INPUTS:
0244 911 :
0244 912 : R3 Event flag number
0244 913 : R4 Address of PCB
0244 914 : R7 Lock mode
0244 915 : R8 Address of LKSB
0244 916 : R9 Flags
0244 917 :
0244 918 : Caller's argument list (offsets from AP)
0244 919
0244 920 : OUTPUTS:
0244 921 :
0244 922 : R0 Completion code
0244 923
0244 924 : IMPLICIT OUTPUTS:
0244 925 :
0244 926 : Caller's lock status block gets final request status (perhaps
0244 927 : asynchronously)
0244 928
0244 929 : COMPLETION CODES:
0244 930 :
0244 931 : In R0:
0244 932 :
0244 933 : $$$_NORMAL Successful completion
0244 934 : $$$_SYNCH Synchronous successful completion
0244 935 : $$$_IVLOCKID Invalid lock id
0244 936 : $$$_ACCVIO Access violation (on resource name)
0244 937 : $$$_PARNOTGRANT Parent lock not granted
0244 938 : $$$_NOSYSLCK No SYSLCK privilege (needed for a system lock)
0244 939 : $$$_IVBUFLN Resource name length = 0 or > 31
0244 940 : $$$_INSFMEM Insufficient dynamic memory
0244 941 : $$$_EXENQLM Exceeded enqueue quota
0244 942 : $$$_NOTQUEUED Request was not queued
0244 943 : $$$_NOPRIV No privilege (to not charge quota)
0244 944 :
0244 945 : In LKSB:
0244 946 :
0244 947 : $$$_NORMAL Successful completion
0244 948 : $$$_DEADLOCK Deadlock detected
0244 949 : $$$_ABORT Lock was dequeued before being granted
0244 950 : --
0244 951
0244 952 .IF NDF LOADSW

```



```
0000004C 953 .PSECT Y$EXEPAGED
004C 954 .ENDC
004C 955
004C 956 .ENABL LSB
004C 957
004C 958 ;
004C 959 ; Errors:
004C 960 ;
004C 961
50 0C 3C 004C 962 ACCVIO: MOVZWL S^#SS$_ACCVIO,R0 ; Access violation on res. name or LKSB
05 11 004F 963 BRB 8$
50 034C 8F 3C 0051 964 2$: MOVZWL #SS$_IVBUFLN,R0 ; Invalid length for resource name
00000543 EF 17 0056 965 8$: JMP ERROR_EXIT_R0
005C 966
005C 967 NEW_LOCK:
005C 968
005C 969 ; Get pointer to resource name, probe it, and check the length
005C 970 ; for legality (0 < length <= RSB$_MAXLEN).
005C 971
50 14 AC D0 005C 972 MOVL RESNAM(AP),R0 ; Get address of resource name descriptor
0060 973 IFNORD #8,(R0),ACCVIO ; Branch if descriptor not readable
5A 60 7D 0066 974 MOVQ (R0),R10 ; Get length (R10) and address (R11)
5A 5A 3C 0069 975 MOVZWL R10,R10 ; Clear top half of length
E3 13 006C 976 BEQL 2$ ; Error, length is zero
006E 977 ASSUME RSB$_MAXLEN LE 512
1F 5A D1 006E 978 CMPL R10,#RSB$_MAXLEN ; Is length > maximum allowed?
DE 1A 0071 979 BGTRU 2$ ; Yes, error, length is too big
0073 980 IFNORD R10,(R11),ACCVIO ; Branch if string not readable
0079 981
0079 982 ; Allocate a lock block.
0079 983 ; NOTE: There is an implicit assumption here (unchecked!) that
0079 984 ; the minimum size of an SRP is 96 bytes.
0079 985
0079 986 ASSUME LKB$_LENGTH LE 96
0079 987 ASSUME LKB$_TYPE EQ LKB$_SIZE+2
0079 988
50 00000000 GF DE 007C 989 SETIPL #IPL$_ASTDEL ; Raise IPL (to allocate pool)
0083 990 MOVAL G^IOC$_GL_SRPFL,R0 ; *** Combine this and the following
0083 991 ; instruction when loading is resolved
52 00 B0 OF 0083 992 REMQUE @ (R0),R2 ; Try lookaside list
10 1C 0087 993 BVC 10$ ; Have one
51 0060 8F 3C 0089 994 MOVZWL #LKB$_LENGTH,R1 ; List empty - do it the hard way
50 D4 008E 995 CLRL R0 ; No cleanup routine needed
00000000 GF 16 0090 996 JSB G^EXE$_ALONPAGWAIT ; Allocate; wait if necessary
BD 50 E9 0096 997 BLBC R0,8$ ; None there and resource wait off
56 52 D0 0099 998 10$: MOVL R2,R6 ; R6 will point to LKB
08 A6 00350060 8F D0 009C 999 MOVL #<DYN$_LKB@16>!, ; Store size and type fields
00A4 1000 LKB$_LENGTH,LKB$_SIZE(R6)
00A4 1001
00A4 1002 ; Fill in various fields in LKB
00A4 1003
00A4 1004 ASSUME ASTPRM EQ ASTADR+4
00A4 1005 ASSUME LKB$_GRMODE EQ LKB$_RQMODE+1
00A4 1006
37 A6 53 90 00A4 1007 MOVB R3,LKB$_EFN(R6) ; Store event flag number
34 A6 57 9B 00A8 1008 MOVZBW R7,LKB$_RQMODE(R6) ; Store req. mode; clear granted mode
24 A6 58 D0 00AC 1009 MOVL R8,LKB$_LKSB(R6) ; Store LKSB address
```

```
1C AC D0 00B0 1010      MOVL   ASTADR(AP),-      ; Store completion AST address
1C A6      00B3 1011      LKBSL CPLASTADR(R6)
24 AC D0 00B5 1012      MOVL   BLKAST(AP),-      ; Store blocking AST address
20 A6      00B8 1013      LKBSL BLKASTADR(R6)
20 AC D0 00BA 1014      MOVL   ASTPRM(AP),-      ; Store AST parameter
14 A6      00BD 1015      LKBSL ASTPRM(R6)
60 A4 D0 00BF 1016      MOVL   PCB$-PID(R4),-      ; Store PID
0C A6      00C2 1017      LKBSL -PID(R6)
4C A6 B4 00C4 1018      CLRW   LKB$W-REFCNT(R6)      ; Clear sub LKB reference count
      00C7 1019
      00C7 1020      ; Allocate a resource block (RSB). This is done now because the
      00C7 1021      ; resource name must be copied from the caller's buffer to a system
      00C7 1022      ; one before it is used. It is copied right into the
      00C7 1023      ; resource block because the common case is that the resource does
      00C7 1024      ; not currently exist. If we later find the resource, this RSB
      00C7 1025      ; will be deallocated.
      00C7 1026
51 50 AA 9E 00C7 1027      MOVAB  RSB$K_LENGTH(R10),R1      ; Add length of name to fixed size
00000000'GF 51 D1 00CB 1028      CMPL  R1,G^IOC$GL_SRPSIZE      ; Will it fit in a SRP?
      25 1A 00D2 1029      BGTRU  15$      ; No
50 00000000'GF DE 00D4 1030      MOVAL  G^IOC$GL_SRPFL,R0      ; *** Combine this and the following
      00DB 1031      ; instruction when loading is resolved
      52 00 B0 0F 00DB 1032      REMQUE  a(R0),R2      ; Try lookaside list
      18 1D 00DF 1033      BVS  15$      ; Didn't get one
      00E1 1034
      00E1 1035 13$:      ; Continue building RSB and LKB
      00E1 1036
      00E1 1037      ASSUME  RSB$B_TYPE EQ RSB$W_SIZE+2
      00E1 1038      ASSUME  RSB$B_DEPTH EQ RSB$B_TYPE+1
      00E1 1039
      08 A2 51 3C 00E1 1040      MOVZWL R1,RSB$W_SIZE(R2)      ; Store size of RSB and zero depth
      58 52 D0 00E5 1041      MOVL  R2,R8      ; R8 will point to RSB
      00E8 1042
      00E8 1043      ; Copy resource name to RSB and branch to non-paged PSECT.
      00E8 1044
      50 A8 57 54 D0 00E8 1045      MOVL  R4,R7      ; Save PCB address
      6B 5A 28 00EB 1046      MOVCL  R10,(R11),RSB$T_RESNAM(R8) ; R0 - R5 not valid anymore
      54 57 D0 00F0 1047      MOVL  R7,R4      ; Restore PCB address
      00000255'EF 17 00F3 1048      JMP  40$      ; Branch to non-paged PSECT
      00F9 1049
      00F9 1050 15$:      ; Need to allocate a RSB from pool because it either won't fit
      00F9 1051      ; in a SRP or the SRP list is empty.
      00F9 1052
50 0000058E'EF 9E 00F9 1053      MOVAB  L^CLEANUP4,R0      ; Set address of cleanup routine
      00000000'GF 16 0100 1054      JSB  G^EXESALONPAGWAIT      ; Allocate. Wait if necessary.
      D8 50 E8 0106 1055      BLBS  R0,13$      ; Have one
      58 D4 0109 1056      CLRL  R8      ; Error, indicate no RSB to deallocate
      0000024C'EF 17 010B 1057      JMP  27$      ; Deallocate LKB and return
      0111 1058
      0111 1059      .IF NDF LOADSW
      00000244 1060      .PSECT LOCKMGR
      0244 1061
      0244 1062
      0244 1063 ;*****
      0244 1064 ;
      0244 1065 ; Errors
      C^44 1066 ;
```

```
0244 1067 ;*****
0244 1068
0244 1069 ; CLEANUP3 errors
0244 1070
50 2134 8F 3C 0244 1071 23$: MOVZWL #SS$ PARNOTGRANT,R0 ; Parent lock not granted
56 53 D0 0249 1072 25$: MOVL R3,R8 ; Restore address of LKB
5B 50 D0 024C 1073 27$: MOVL R0,R11 ; Save completion code
0331 30 024F 1074 BSBW CLEANUP3 ; Deallocate LKB and RSB; then exit
02EB 31 0252 1075 BRW ERROR_EXIT_R11
0255 1076
0255 1077
0255 1078 40$: ; Get parent id and convert to parent LKB if non-zero. If a parent is
0255 1079 ; specified get parent RSB address (in R7) and get resource access mode
0255 1080 ; from parent RSB. Otherwise, R7 = 0 so we get resource access mode
0255 1081 ; from argument list (maximized with caller's access mode, of course).
0255 1082 ; Note that we raise to IPL$ SYNCH here. After this is performed,
0255 1083 ; there can be no further references to the caller's address space.
0255 1084
0255 1085 ASSUME LKB$K_GRANTED GT 0
0255 1086 ASSUME LKB$K_CONVERT EQ 0
0255 1087 ASSUME RSB$B_RMOD EQ RSB$W_GROUP+2
0255 1088
53 56 D0 0255 1089 MOVL R6,R3 ; Save LKB address
56 57 D4 0258 1090 CLRL R7 ; Assume no parent RSB
56 18 AC D0 025A 1091 MOVL PARID(AP),R6 ; Get parent id
23 13 025E 1092 BEQL 45$ ; Branch if no parent specified
51 56 D0 0260 1093 SETIPL #IPL$ SYNCH ; Raise IPL
0703 30 0263 1094 MOVL R6,R1 ; Move parent id
DD 50 E9 0266 1095 BSBW VERIFYPARLOCKID ; Get LKB in R6 and caller's access mode
36 A6 95 0269 1096 BLBC R0,25$ ; in R1. Branch if error
05 18 026C 1097 TSTB LKB$B_STATE(R6) ; Is parent lock in grant or cvt state?
01 E1 026F 1098 BGEQ 43$ ; Yes
CE 28 A6 0271 1099 BBC #LCK$V_CONVERT,- ; No, but if CONVERT bit is set, then
0273 1100 LKB$W_FLAGS(R6),23$ ; it's okay as lock is in a transient
0276 1101 ; convert state. Otherwise, error!
57 50 A6 D0 0276 1102 43$: MOVL LKB$L_RSB(R6),R7 ; Yes, get parent RSB address
50 4C A7 D0 027A 1103 MOVL RSB$W_GROUP(R7),R0 ; Get parent's group and res. acmode
4C A6 B6 027E 1104 INCW LKB$W_REFCNT(R6) ; Increment parent's sub LKB ref. count
1C 11 0281 1105 BRB 50$
0283 1106
0283 1107 45$: ; No parent LKB so get specified access mode and maximize with
0283 1108 ; caller's access mode.
0283 1109
50 DC 0283 1110 MOVPSL R0 ; Read current PSL
51 50 16 EF 0285 1111 EXTZV #PSL$V_PRVMOD,- ; Extract previous mode field
28 AC 50 02 0287 1112 #PSL$S_PRVMOD,R0,R1
8B 8F 028A 1113 BICB3 #^C<3>,ACMODE(AP),R0 ; Get specified access mode
50 51 91 0290 1114 SETIPL #IPL$ SYNCH ; Raise IPL
50 03 1F 0293 1115 CMPB R1,R0 ; Compare with caller's access mode
50 50 51 90 0296 1116 BLSSU 47$ ; Use specified access mode (R0)
50 50 10 78 0298 1117 MOVB R1,R0 ; Use caller's access mode (R1)
029B 1118 47$: ASHL #16,R0,R0 ; Move to bits <16:23>
029F 1119
029F 1120 50$: ; Store parent LKB address or 0 (in R6). Then store caller's
029F 1121 ; access mode with NODELETE bit set (caller's access mode is in R1,
029F 1122 ; resource access mode is in R0 <16:23>).
029F 1123
```

			029F	1124	ASSUME	LKBSB_RMOD	EQ	ACBSB_RMOD	
			029F	1125	ASSUME	LKBSM_NODELETE	EQ	ACBSM_NODELETE	
			029F	1126					
48	A3	56	D0	029F	1127	MOVL	R6,LKBSL_PARENT(R3)		; Store parent LKB ptr in new LKB
	56	53	D0	02A3	1128	MOVL	R3,R6		; Restore LKB address
OB	A6	51	89	02A6	1129	BISB3	#LKBSM_NODELETE,R1,-		; Store access mode in LKB and set
				02AB	1130		LKBSB_RMOD(R6)		; no delete bit
	55	51	D0	02AB	1131	MOVL	R1,R5-		; Move access mode
				02AE	1132				
				02AE	1133				; R0 <16:23> contains resource access mode; R0 <0:15> contains
				02AE	1134				; parent's group if there is a parent. R5 contains
				02AE	1135				; caller's access mode; R7 contains parent RSB address or 0.
				02AE	1136				; Store composite group number and access mode in RSB.
				02AE	1137				
				02AE	1138				
				02AE	1139	ASSUME	RSBSW_GROUP	EQ	RSBSL_PARENT+4
				02AE	1140	ASSUME	RSBSB_RMOD	EQ	RSBSW_GROUP+2
				02AE	1141				
	51	50	D0	02AE	1141	MOVL	R0,R1		; Move composite fields
	50	57	D0	02B1	1142	MOVL	R7,R0		; Move parent RSB address
		17	12	02B4	1143	BNEQ	53\$		; Store if this is a sub-lock
	59	10	B3	02B6	1144	BITW	#LCKSM_SYSTEM,R9		; Is this a system name?
		07	12	02B9	1145	BNEQ	52\$		; Yes
51	00BE	C4	B0	02BB	1146	MOVW	PCBSW_GRP(R4),R1		; Group name - store group number
		0B	11	02C0	1147	BRB	53\$		
	01	55	91	02C2	1148	CMPB	R5,#PSLSC_EXEC		; System name - allow without priv. if
		06	1B	02C5	1149	BLEQU	53\$		; caller is from EXEC or KERNEL mode
				02C7	1150	IFNPRIV	SYSLCK,61\$		; SUPER or USER mode needs privilege
48	A8	50	7D	02CD	1151	MOVQ	R0,RSBSL_PARENT(R8)		; Store parent, group, acmode in RSB
4F	A8	5A	90	02D1	1152	MOVB	R10,RSBSB_RSNLEN(R8)		; Store resource name length in RSB
	2A	A6	B4	02D5	1153	CLRW	LKBSW_STATUS(R6)		; Clear status bits
				02D8	1154				
				02D8	1155				; See if any special operations must be performed.
				02D8	1156				; The caller's mode is in R5. The PCB address is in R4.
				02D8	1157				
59	01E0	8F	B3	02D8	1158	BITW	#LCKSM_RECOVER-		; Any special bits set?
				02DD	1159		!LCKSM_PROTECT-		
				02DD	1160		!LCKSM_NOQUOTA-		
				02DD	1161		!LCKSM_CVTSYS,R9		
		3E	13	02DD	1162	BEQL	59\$		; No
				02DF	1163				
				02DF	1164				; If RECOVER bit is set, the verify process has privilege to set
				02DF	1165				; it and if so, also set the PROTECT and NOQUEUE bits.
				02DF	1166				
0A	59	07	E1	02DF	1167	BBC	#LCKSV_RECOVER,R9,54\$		; Branch if RECOVER is not set
		1A	E1	02E3	1168	BBC	#PCBSV_RECOVER,-		; Branch if no privilege
	4F	24	A4	02E5	1169		PCBSL_SYS(R4),60\$		
59	0104	8F	A8	02E8	1170	BISW	#LCKSM_NOQUEUE!LCKSM_PROTECT,R9		; Set related bits
				02ED	1171				
				02ED	1172				; If PROTECT is set, then set corresponding bit in status
				02ED	1173				
				02ED	1174				
06	59	08	E1	02ED	1174	BBC	#LCKSV_PROTECT,R9,55\$		; Branch if PROTECT is not set
	0200	8F	A8	02F1	1175	BISW	#LKBSM_PROTECT,-		; Set PROTECT bit in status
		2A	A6	02F5	1176		LKBSW_STATUS(R6)		
				02F7	1177				
				02F7	1178				; If CVTSYS is set, then verify caller is from EXEC or KERNEL
				02F7	1179				; mode and that parent lock is system owned. Then also set
				02F7	1180				; NOQUEUE and SYNCSTS bits and clear PID.

```
13 59 06 E1 02F7 1181
50 48 A6 D0 02F7 1182 BBC #LCK$V CVTSYS,R9,57$ ; Branch if CVTSYS is not set
05 13 02FB 1183 MOVL LKB$$_PARENT(R6),R0 ; Get parent LKB address
0C A0 D5 0301 1184 BEQL 56$ ; No parent
23 12 0304 1185 TSTL LKB$$_PID(R0) ; Is parent system owned?
59 0C A8 0306 1186 BNEQ 60$ ; No, error
0C A6 D4 0309 1187 56$: BISW #LCK$M SYNCSTS:LCK$M_NOQUEUE,R9 ; Set related bits
04 11 030C 1188 CLRL LKB$$_PID(R6) ; Clear PID
030E 1189 BRB 58$ ; Do access mode check
030E 1190
030E 1191 57$: ; If NOQUOTA is set, then verify caller is from EXEC or KERNEL
030E 1192 ; mode and set internal NOQUOTA bit.
030E 1193
0B 59 05 E1 030E 1194 BBC #LCK$V NOQUOTA,R9,59$ ; Branch if NOQUOTA is not set
01 55 91 0312 1195 58$: CMPB R5,#PSC$_EXEC ; Is access mode EXEC or KERNEL?
20 1A 0315 1196 BGTRU 62$ ; No - error
20 A8 0317 1197 BISW #LKB$M_NOQUOTA,- ; Yes, set NOQUOTA status
2A A6 0319 1198 LKB$$_STATUS(R6)
63 11 031B 1199 BRB 85$
031D 1200
54 0080 C4 D0 031D 1201 59$: MOVL PCB$$_JIB(R4),R4 ; Get pointer to JIB
4C A4 B7 0322 1202 DECW JIB$$_ENQCNT(R4) ; Decrement enqueue count remaining
59 18 0325 1203 BGEQ 85$ ; Quota ok
1D 11 0327 1204 BRB 67$ ; No quota - error
0329 1205
0329 1206 ;*****
0329 1207 ;
0329 1208 ; Error branches and out of line code
0329 1209 ;
0329 1210 ;*****
0329 1211
0329 1212 ; CLEANUP2 errors
0329 1213
5B 225C 8F 3C 0329 1214 60$: MOVZWL #SS$_PARNOTSYS,R11 ; Parent not system owned
0A 11 032E 1215 BRB 64$
5B 28F4 8F 3C 0330 1216 61$: MOVZWL #SS$_NOSYSLCK,R11 ; No privilege for system lock
03 11 0335 1217 BRB 64$
5B 24 3C 0337 1218 62$: MOVZWL S^#SS$_NOPRIV,R11 ; No privilege
023B 30 033A 1219 64$: BSBW CLEANUP2 ; Cleanup
0F 11 033D 1220 BRB 69$
033F 1221
033F 1222 ; CLEANUP1 errors
033F 1223
5B 0E32 8F 3C 033F 1224 66$: MOVZWL #SS$_RETRY,R11 ; Retry operation
05 11 0344 1225 BRB 68$
5B 2A44 8F 3C 0346 1226 67$: MOVZWL #SS$_EXENQLM,R11 ; Exceeded enqueue quota
0216 30 034B 1227 68$: BSBW CLEANUP1 ; Cleanup (deallocate LKB, RSB, etc.)
034E 1228
01EF 31 034E 1229 69$: BRW ERROR_EXIT_R11
0351 1230
0351 1231 ; OUT OF LINE CODE
0351 1232
0351 1233 70$: ; We are stalling some lock requests. R0 (low byte) contains
0351 1234 ; stall flag. See if this request should be stalled. Stall values
0351 1235 ; are:
0351 1236 ;
0351 1237 ; -1 Stall all requests
```

```
0351 1238      ; +1      Stall only protected locks (not being recovered)
0351 1239      ; +2      Stall protected locks and root locks
0351 1240
02  50 19 0351 1241      BLSS      74$      ; We are stalling all requests
      05 91 0353 1242      CMPB      R0,#2      ; Are we stalling root locks?
      48 A6 D5 0356 1243      BLSS      72$      ; No
      0D 13 0358 1244      TSTL      LKB$$_PARENT(R6) ; Yes, is this a root lock?
2C 59 08 E1 035B 1245      BEQL      74$      ; Yes, stall this request
28 59 07 E0 035D 1246 72$:      BBC      #LCK$$_PROTECT,R9,90$ ; Don't stall unprotected locks
      1A E0 0361 1247      BBS      #LCK$$_RECOVER,R9,90$ ; Don't stall recovering a lock
      D5 24 A4 0365 1248      BBS      #PCB$$_RECOVER,- ; Don't stall recovery process,
      01F7 30 0367 1249      PCB$$_STS(R4),66$ ; return error instead
      0920 31 036A 1250 74$:      BSBW      CLEANUP1 ; Cleanup
      036D 1251      BRW      STALL_REQ ; Stall this request
      0370 1252
      0370 1253 75$:      ; No lockids. Try extending table
      0370 1254
50  0564'CF 9E 0370 1255      MOVAB     W^CLEANUP1,R0 ; Address of cleanup routine
      0944 30 0375 1256      BSBW      LCK$$_EXTEND_IDTBLW ; Try extending table
      12 50 E8 0378 1257      BLBS      R0,90$ ; Success
      5B 50 D0 037B 1258      MOVL      R0,R11 ; Failure, move status
      CB 11 037E 1259      BRB      68$ ; Cleanup
      0380 1260
      0380 1261 ;*****
      0380 1262 ;
      0380 1263 ; End error branches and out of line code
      0380 1264 ;
      0380 1265 ;*****
      0380 1266
      0380 1267 85$:      ; Store flags and stall this lock request, if necessary.
      0380 1268
      0380 1269      MOVW      R9,LKB$$_FLAGS(R6) ; Store flags
50  28 A6 59 B0 0384 1270      MOVW      G^LCK$$_STALLREQS,R0 ; Are we stalling requests?
      00000000'GF 90 038B 1271      BNEQ      70$ ; Yes
      C4 12 038D 1272
      038D 1273 90$:      ; Allocate a lock id and point the id slot to this LKB.
      038D 1274
      038D 1275      MOVL      G^LCK$$_GL_NXTID,R0 ; Get next lock id
      0394 1276      BEQL      75$ ; No more - try expanding table
51  00000000'GF D0 0396 1277      MOVL      G^LCK$$_GL_IDTBL,R1 ; Get address of lock id table. *** May
      30 A6 50 B0 039D 1278 ; combine with next instr. if no loading
      51 6140 DE 03A1 1280 ; Store lockid index
      00000000'GF 61 3C 03A5 1281 ; Get address of lockid table entry
      32 A6 02 A1 B0 03AC 1282 ; Update ptr to next free id
      61 56 D0 03B1 1283 ; Store lockid sequence number
      03B4 1284 ; Store LKB address in table entry
      03B4 1285 ;
      03B4 1286 ; Now hash resource name and search hash table for a matching
      03B4 1287 ; name and parent RSB address.
      03B4 1288
      03B8 1289      MOVAB     RSB$$_PARENT(R8),R4 ; Point to parent, group, resnam, etc.
      01DE 30 03BB 1290      ADDL      #8,R10 ; Account for parent RSB, group, etc.
      72 50 E8 03BE 1291      BSBW      LCK$$_HASH_SEARCH ; Hash and search the table for a match
      03C1 1292 ; Didn't find one
      03C1 1293 ; Found one - fall through to ...
      03C1 1294
```

SYSENQDEQ
V04-000

J 16
- ENQUEUE/DEQUEUE SYSTEM SERVICES 16-SEP-1984 02:02:16 VAX/VMS M. ro V04-00
NEW_LOCK - New lock request (not convers 5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1
03C1 1295 .DSABL LSB

Page 28
(9)

```
0000 03C1 1297      .IF NDF LOADSW
03C1 1298      .PSECT LOCKMGR
03C1 1299      .ENDC
03C1 1300
03C1 1301 OLD_RESOURCE:
03C1 1302      ; Found a matching resource block (RSB). Address of RSB is in R5.
03C1 1303      ; Address of LKB is in R6. First deallocate the temporary RSB
03C1 1304      ; pointed to by R8. If anyone is waiting
03C1 1305      ; (in either the waiting queue or the conversion queue), then
03C1 1306      ; this request must also wait. If no one is waiting, then this
03C1 1307      ; lock can be granted if it is compatible with the group grant mode.
03C1 1308
50 58 D0 03C1 1309      MOVL R8,R0
00000000 GF 16 03C4 1310      JSB G^EXES$DEANONPAGED      ; Deallocate temporary RSB
58 55 D0 03CA 1311      MOVL R5,R8      ; Use R8 from now on for RSB address
50 A6 58 D0 03CD 1312      MOVL R8,LKB$$_RSB(R6)      ; Store RSB pointer in LKB
03D1 1313
03D1 1314      ; If this resource is being handled remotely, then send a remote
03D1 1315      ; lock request. Otherwise do it here.
03D1 1316
53 38 A8 D0 03D1 1317      MOVL RSB$_CSID(R8),R3      ; Get CSID of system managing this
56 12 03D5 1318      BNEQ REM_LOCK      ; resource and branch if it's not us
03D7 1319
03D7 1320 LCK$LOCAL_LOCK::
03D7 1321      ; Return here to handle locks locally after a directory lookup.
03D7 1322
03D7 1323      ASSUME RSB$_WTQFL EQ RSB$_CVTQFL+8
03D7 1324
00000002 00000000 GF D6 03D7 1325      .IF NE CAS$ MEASURE
03D7 1326      INCL G^PMSS$GL_ENQNEW_LOC
03DD 1327      .ENDC
03DD 1328
50 18 A8 E 03DD 1329      MOVAL RSB$_CVTQFL(R8),R0      ; Get address of conversion queue
60 50 D1 03E1 1330      CMPL R0,(R0)      ; Queue empty?
50 2A 12 03E4 1331      BNEQ 70$      ; No
50 08 C0 03E6 1332      ADDL #8,R0      ; Get address of wait queue
51 50 D0 03E9 1333      MOVL R0,R1      ; Save in R1
51 60 D1 03EC 1334 60$:      CMPL (R0),R1      ; Is this the end of the queue?
15 12 03EF 1335      BNEQ 65$      ; No
03F1 1336
03F1 1337 62$:      ; No waiting requests (or RECOVER bit is set); is the lock compatible?
03F1 1338
51 34 A6 9A 03F1 1339      MOVZBL LKB$_RQMODE(R6),R1      ; Get lock mode
55 0C A3 9A 03F5 1340      MOVZBL RSB$_GGMODE(R8),R5      ; Get group grant mode
14 FC01 CF45 51 E1 03F9 1341      BBC R1,LCK$COMPAT_TBL[R5],72$ ; Branch if incompatible
0400 1342
0400 1343      ; Lock can be granted
0400 1344
0208 30 0400 1345      BSBW LCK$GRANT_LOCK      ; Returns status n R0
00BD 31 0403 1346      BRW LCK$SYNC_EXIT
0406 1347
0406 1348 65$:      ; Skip this lock if it's in a SCS wait state (anything but
0406 1349      ; LKB$_WAITING).
0406 1350
50 60 D0 0406 1351      MOVL (R0),R0      ; Point to lock
FF 8F 91 0409 1352      CMPB #LKB$_WAITING,-      ; Is it waiting?
FE A0 040C 1353      LKB$_STATE-LKB$_SQFL(R0)
```



```

      DC 12 040E 1354      BNEQ 60$      ; No, skip over it
           0410 1355
           0410 1356 70$:  ; Request cannot be granted due to other locks in the waiting
           0410 1357      ; or conversion queue ahead of us. If RECOVER bit is set,
           0410 1358      ; then ignore these waiters and try granting anyway.
           0410 1359
      DD 59 07 E0 0410 1360      BBS #LCK$V_RECOVER,R9,62$ ; Branch if recovering a lock
           C414 1361
           0414 1362 72$:  ; Request cannot be granted due to other waiters or incompatibility.
           0414 1363      ; The request gets queued unless the user requested that it not
           0414 1364      ; be queued if it cannot be granted.
           0414 1365
      OD 59 02 E0 0414 1366      BBS #LCK$V_NOQUEUE,R9,80$ ; Br. if request should not be queued
54 00000000'GF D0 0418 1367      MOVL G^SCH$GL_CURPCB,R4 ; Get PCB address
           03A6 30 041F 1368      BSBW LCK$QUEUEWAIT ; Insert lock on wait queue, etc.
           00D3 31 0422 1369      BRW LCK$QUEUED_EXIT ; Exit system service
           0425 1370
           0425 1371 80$:  ; Request is not to be queued. Clean up and return status in R0.
           0425 1372
      SB 09B8 8F 3C 0425 1373      MOVZWL #SS$ NOTQUEUED,R11 ; Status
           00E0 31 042A 1374      BRW LCK$NOT_QUEUED
           042D 1375
           042D 1376 REM_LOCK:
           042D 1377      ; Send this lock request to another system
           042D 1378
      00000000'GF 17 042D 1379      JMP G^LCK$SND_LOCKREQ ; Send remote lock request
           0433 1380
           0433 1381      ; Possible return entry points are:
           0433 1382      ;
           0433 1383      ; LCK$LOCAL_LOCK Handle request here after all
           0433 1384      ; LCK$SYNC_EXIT Lock request was granted
           0433 1385      ; LCK$QUEUED_EXIT Lock request was queued
           0433 1386      ; LCK$NOT_QUEUED Lock request was not queued
```

```
00000433 1388      .IF NDF LOADSW
0433 1389      .PSECT LOCKMGR
0433 1390      .ENDC
0433 1391
0433 1392 NEW_RESOURCE:
0433 1393      ; Resource does not exist, so create it. Register usage is:
0433 1394      ;
0433 1395      ; R6 Address of LKB
0433 1396      ; R7 Address of parent RSB (or 0 if no parent)
0433 1397      ; R8 Address of RSB being created
0433 1398      ; R9 Flags
0433 1399      ; R11 Address of last RSB in hash chain
0433 1400
50 0A 36 90 0433 1401 85$: MOVB #DYN$C_RSB,- ; Store type field
0A A8 58 D0 0435 1402      RSB$B_TYPE(R8)
0437 1403      MOVL R8,LKB$$_RSB(R6) ; Store RSB pointer in LKB
043B 1404
043B 1405      ; Insert RSB into hash chain. R11 points to previous entry
043B 1406      ; which was, until now, the last one in the chain.
043B 1407
043B 1408      MOVL R8,RSB$$_HSHCHN(R11) ; Make previous entry point to this one
04 6B 58 D0 043E 1409      CLRL RSB$$_HSHCHN(R8) ; This one now ends the chain
0A A8 5B D0 0440 1410      MOVL R11,RSB$$_HSHCHNBK(R8) ; Back pointer points to previous one
0444 1411
0444 1412      ; Fill in remaining fields in RSB
0444 1413
0444 1414      ASSUME RSB$$_REFCNT EQ RSB$$_VALUEQNUM+4
0444 1415      ASSUME RSB$$_BLKASTCNT EQ RSB$$_REFCNT+2
0444 1416      ASSUME RSB$$_CVTQFL EQ RSB$$_GRQFL+8
0444 1417      ASSUME RSB$$_WTQFL EQ RSB$$_CVTQFL+8
0444 1418      ASSUME RSB$$_CGMODE EQ RSB$$_GGMODE+1
0444 1419      ASSUME RSB$$_STATUS EQ RSB$$_CGMODE+1
0444 1420
50 10 A8 DE 0444 1421      MOVAL RSB$$_GRQFL(R8),R0 ; Initialize all three queue headers
51 50 D0 0448 1422      MOVL R0,R1
81 50 D0 044B 1423      MOVL R0,(R1)+ ; Granted queue
81 80 7E 044E 1424      MOVAQ (R0)+(R1)+
81 50 D0 0451 1425      MOVL R0,(R1)+ ; Conversion queue
81 80 7E 0454 1426      MOVAQ (R0)+(R1)+
81 50 D0 0457 1427      MOVL R0,(R1)+ ; Waiting queue
61 50 D0 045A 1428      MOVL R0,(R1)
28 A8 7C 045D 1429      CLRQ RSB$$_VALBLK(R8) ; Clear value block
30 A8 7C 0460 1430      CLRQ RSB$$_VALBLK+8(R8)
3C A8 7C 0463 1431      CLRQ RSB$$_VALUEQNUM(R8) ; Clear value block sequence number,
0466 1432      ; sub-RSB reference count and
0466 1433      ; blocking AST count
0C A8 D4 0466 1434      CLRL RSB$$_GGMODE(R8) ; Clear modes, status
46 A8 B4 0469 1435      CLRW RSB$$_REQSEQNM(R8) ; Clear req. seq. number
046C 1436
046C 1437      ; If there is a parent RSB then incr. sub-RSB reference count
046C 1438      ; and check for maximum depth of resource name tree.
046C 1439
046C 1440      TSTL R7 ; Is there a parent?
046E 1441      BEQL 90$ ; No parent
0B A7 01 81 0470 1442      ADDB3 #1,RSB$$_DEPTH(R7),- ; Our depth is 1 more than our
0B A8 0474 1443      RSB$$_DEPTH(R8) ; parent's depth
0B A8 91 0476 1444      CMPB RSB$$_DEPTH(R8),- ; Is our depth equal to
```

```
00000000'GF      0479 1445      G^LCK$GB_MAXDEPTH      ; maximum allowed?
                  1E 047E 1446      BGEQU 88$           ; Yes - error
                  40 A7 B6 0480 1447      INCW RSB$W_REFCNT(R7) ; Increment parent's sub RSB ref. count
                  38 A7 D0 0483 1448      MOVL RSB$L_CSID(R7),- ; Our parent's CSID becomes
                  38 A8      0486 1449      RSB$L_CSID(R8)      ; ours also
                  2C 13 0488 1450      BEQL 95$           ; Resource is managed here
53 38 A8 D0 048A 1451      MOVL RSB$L_CSID(R8),R3      ; Get CSID of destination system
                  9D 11 048E 1452 87$: BRB REM_LOCK      ; Resource is managed by another system
                  0071 31 0490 1453 88$: BRW DEPTH_ERROR
                  0493 1454
                  0493 1455 90$: ; This resource has no parent. Send lock request to appropriate
                  0493 1456      ; directory system. If this system is the directory system for this
                  0493 1457      ; resource, then turn RSB into a directory entry
                  0493 1458
53 00000000'GF D0 0493 1459      MOVL G^LCK$GL_DIRVEC,R3      ; Get address of directory vector
                  10 13 049A 1460      BEQL 93$           ; No vector, we are dir. sys.
                  51 44 A8 3C 049C 1461      MOVZWL RSB$W_HASHVAL(R8),R1 ; Get hash value
                  52 D4 04A0 1462      CLRL R2           ; Clear high order hash value
51 50 51 F4 A3 7B 04A2 1463      EDIV -12(R3),R1,R0,R1 ; Remainder in R1
                  53 6341 D0 04A8 1464      MOVL (R3)[R1],R3 ; Get directory system CSID
                  38 A8 53 D0 04AC 1465 93$: MOVL R3,RSB$L_CSID(R8) ; Store it in RSB
                  DC 12 04B0 1466      BNEQ 87$           ; Not us - do a directory lookup
                  01 A8 04B2 1467      BISW #RSB$M_DIRENTRY,- ; Set directory entry bit
                  0E A8 04B4 1468      RSB$W_STATUS(R8)
                  04B6 1469
                  51 34 A6 9A 04B6 1470 95$: MOVZBL LKB$B_RQMODE(R6),R1 ; Get requested lock mode
                  0153 30 04BA 1471      BSBW LCK$GRANT_LOCK_ALT ; Grant lock
                  04BD 1472
                  00000002 04BD 1473      .IF NE CAS$ MEASURE
00000000'GF D6 04BD 1474      INCL G^PM$SGL_ENQNEW_LOC
                  04C3 1475      .ENDC
                  04C3 1476
                  04C3 1477 LCK$SYNC_EXIT::
                  04C3 1478      ; The request has been satisfied synchronously. Status is already
                  04C3 1479      ; in R0 and LKB$L_LKST1. Insert the lock onto the head of the
                  04C3 1480      ; lock list in the PCB (unless it's a system lock) and store
                  04C3 1481      ; value block, if specified.
                  04C3 1482      ; Note: Conversions should not use this exit path as they are
                  04C3 1483      ; already on the PCB's lock list.
                  04C3 1484
                  04C3 1485      ASSUME LCK$V_VALBLK EQ 0
                  04C3 1486
                  0C A6 D5 04C3 1487      TSTL LKB$L_PID(R6) ; Is this a system owned lock?
                  OD 13 04C6 1488      BEQL 10$           ; Yes, don't insert onto PCB queue
54 00000000'GF D0 04C8 1489      MOVL G^SCH$SGL_CURPCB,R4 ; Get address of PCB
                  40 A6 OE 04CF 1490      INSQUE LKB$L_OWNOFL(R6),- ; Insert lock at head of process
                  0104 C4 04D2 1491      PCB$L_LOCKQFL(R4) ; lock list
                  23 59 E9 04D5 1492 10$: BLBC R9,LCK$NORET_VALBLK ; Branch if no value block
                  04D8 1493
                  04D8 1494 LCK$RET_VALBLK::
                  04D8 1495      ; Store RSB's value block in caller's value block, store LKSB,
                  04D8 1496      ; and return. Status is already in R0 and LKB$L_LKST1.
                  04D8 1497      ; However, if the value block is marked invalid, then we will change
                  04D8 1498      ; the status in LKB$L_LKST1 to SS$_VALNOTVALID.
                  04D8 1499
                  04D8 1500      SETIPL #IPL$_ASTDEL ; Lower IPL to write in caller's
                  04DB 1501      ; address space but still block ASTs
```

```

      02 B3 04DB 1502      BITW  #RSBSM_VALINVLD,-      ; Is value block valid?
      0E A8      04DD 1503      RSB$W_STATUS(R8)
      06 13 04DF 1504      BEQL  10$      ; Yes
09F0 8F B0 04E1 1505      MOVW  #SS$ VALNOTVALID,-      ; No, change completion status
      2C A6      04E5 1506      LKB$C_LKST1(R6)
51 24 A6 D0 04E7 1507 10$:      MOVL  LKB$L_LKSB(R6),R1      ; Get LKSB address
81 2C A6 7D 04EB 1508      MOVQ  LKB$L_LKST1(R6),(R1)+      ; Store LKSB
      28 A8 7D 04EF 1509      MOVQ  RSB$Q_VALBLK(R8),(R1)+      ; Copy RSB value block to caller's
      30 A8 7D 04F3 1510      MOVQ  RSB$Q_VALBLK+8(R8),(R1)+      ; value block
      04      04F7 1511      RET      ; Return
      04F8 1512
      04F8 1513
      04F8 1514
      04F8 1515 LCK$QUEUED_EXIT::
      04F8 1516      ; Come here if request was queued.
      04F8 1517
50 01 3C 04F8 1518      MOVZWL S^#SS$_NORMAL,R0      ; Indicate it was queued successfully
      04FB 1519
      04FB 1520 LCK$NORET_VALBLK::
      04FB 1521      ; Exit service with status in R0. Copy contents of internal
      04FB 1522      ; LKSB to the caller's LKSB. Remember, once IPL is lowered to 0
      04FB 1523      ; the LKB can be instantly deleted out from under us.
      04FB 1524
      04FB 1525      SETIPL #IPL$_ASTDEL      ; Lower IPL to write in caller's
      04FE 1526      ; address space but still block ASTs
      2C A6 7D 04FE 1527      MOVW  LKB$L_LKST1(R6),-      ; Store contents of LKSB
      24 B6      0501 1528      @LKB$C_LKSB(R6)
      04      0503 1529      RET      ; Return
      0504 1530
```

```
0504 1532      .SBTTL Error Handling for $ENQ
0504 1533
0504 1534 :++
0504 1535 : FUNCTIONAL DESCRIPTION:
0504 1536 :
0504 1537 :     This routine performs the error handling for the $ENQ system
0504 1538 :     service. This routine has a number of entry points, depending
0504 1539 :     on what the error is. Each error entry point backs up the
0504 1540 :     operations performed thus far. Consequently, the order of operations
0504 1541 :     in this routine must be exactly the reverse of the order of
0504 1542 :     operations in the $ENQ system service.
0504 1543 :
0504 1544 : CALLING SEQUENCE:
0504 1545 :
0504 1546 :     Branch to error entry point. This routine returns from the system
0504 1547 :     service. The entry points named CLEANUPx are called with a BSBW
0504 1548 :     and they return to the caller. Then the appropriate ERROR_EXIT
0504 1549 :     may be branched to.
0504 1550 :
0504 1551 : INPUT PARAMETERS:
0504 1552 :
0504 1553 :     R0      Completion code (some entry points)
0504 1554 :     R6      Address of LKB (some entry points)
0504 1555 :     R8      Address of RSB (some entry points)
0504 1556 :     R11     Completion code (some entry points)
0504 1557 :             Address of previous RSB (DEPTH_ERROR entry point only)
0504 1558 :
0504 1559 : OUTPUT PARAMETERS:
0504 1560 :
0504 1561 :     R0      Status code
0504 1562 :
0504 1563 : COMPLETION CODES:
0504 1564 :
0504 1565 :     $$$_ACCVIO      Access violation (on LKSB or resource name)
0504 1566 :     $$$_BADPARAM    Bad lock mode
0504 1567 :     $$$_IVLOCKID    Invalid lock id
0504 1568 :     $$$_CVTUNGRANT  Attempted to convert an ungranted lock
0504 1569 :     $$$_PARNOTGRANT Parent lock not granted
0504 1570 :     $$$_NOSYSLCK    No SYSLCK privilege (needed for a system lock)
0504 1571 :     $$$_IVBUFLN     Resource name length = 0 or > 31
0504 1572 :     $$$_INSFMEM     Insufficient dynamic memory
0504 1573 :     $$$_EXASTLM     Exceeded AST quota
0504 1574 :     $$$_EXENQLM     Exceeded enqueue quota
0504 1575 :     $$$_NOTQUEUED   Request was not queued
0504 1576 :     $$$_EXDEPTH     Exceeded allowed depth of resource name tree
0504 1577 :
0504 1578 : SIDE EFFECTS:
0504 1579 :
0504 1580 :     None
0504 1581 : --
0504 1582 :
0504 1583 :     .IF NDF LOADSW
00000504 1584 :     .PSECT LOCKMGR
0504 1585 :     .ENDC
0504 1586 :
0504 1587 :     .ENABL LSB
0504 1588
```

```
0504 1589 DEPTH_ERROR:
0504 1590 ; Remove RSB from hash chain and deallocate both LKB (R6)
0504 1591 ; and RSB (R8). R11 points to previous RSB in hash chain.
0504 1592
0504 1593 CLRL RSB$L_HSHCHN(R11) ; End hash chain one RSB earlier
SB OE1A 6B D4 0506 1594 MOVZWL #SS$_EXDEPTH,R11 ; Store status
08 11 050B 1595 BRB 10$
050D 1596
050D 1597 LCK$NOT_QUEUED::
050D 1598 ; Deallocate lock id. LKB address in R6, status in R11.
050D 1599
00000002 050D 1600 .IF NE CAS MEASURE
00000000'GF D6 050D 1601 INCL G^PR$$_GL_ENQNOTQD
0513 1602 .ENDC
0513 1603
0513 1604 CLRL R8 ; Indicates no RSB to deallocate
0515 1605
0515 1606 10$: MOVZWL LKB$L_LKID(R6),R0 ; Get lock id index
51 00000000'GF D0 0519 1607 MOVL G^LCK$$_GL_IDTBL,R1 ; *** Combine with next instr.
51 6140 DE 0520 1608 MOVAL (R1)[R0],R1 ; Point to table entry
00000000'GF B0 0524 1609 MOVW G^LCK$$_GL_NXTID,(R1) ; Store next id in this id's slot
02 A1 32 A6 01 A1 052B 1610 ADDW3 #1,LKB$L_LKID+2(R6),2(R1) ; Incr. and store sequence number
04 1C 0531 1611 BVC 20$ ; Didn't overflow to a system address
02 A1 01 B0 0533 1612 MOVW #1,2(R1) ; Overflowed - restart seq. number at 1
00000000'GF 50 D0 0537 1613 20$: MOVL R0,G^LCK$$_GL_NXTID ; This id becomes the next one
24 10 053E 1614 BSBB CLEANUP1 ; Cleanup (puts PCB address in R4)
0540 1615
0540 1616 ERROR_EXIT_R11:
50 5B D0 0540 1617 MOVL R11,R0 ; Move status to R0
0543 1618
0543 1619 ERROR_EXIT_R0:
0543 1620 ; Everything has been cleaned up; status is in R0. Set event flag
0543 1621 ; and exit.
0543 1622
0543 1623 SETIPL #IPL$_ASTDEL ; Lower IPL
0546 1624 PUSHL R0 ; Save status
0548 1625 MOVL G^SCH$$_GL_CURPCB,R4 ; Get PCB address
51 60 A4 D0 054F 1626 MOVL PCB$L_PID(R4),R1 ; Get PID
52 02 3C 0553 1627 MOVZWL #PRI$_RESAVL,R2 ; Get priority increment class
53 04 AC 9A 0556 1628 MOVZBL EFN(AP),R3 ; Get event flag
00000000'GF 16 055A 1629 JSB G^SCH$$_POSTEF ; Set event flag
50 8ED0 0560 1630 POPL R0 ; Restore status
04 0563 1631 RET
0564 1632
0564 1633
0564 1634 ; Cleanup subroutine. This subroutine has various entry points which
0564 1635 ; are called depending on how much cleanup is required.
0564 1636
0564 1637 : Inputs:
0564 1638 :
0564 1639 : R6 Address of LKB
0564 1640 : R8 Address of RSB or 0 to indicate no RSB
0564 1641 :
0564 1642 : Outputs:
0564 1643 :
0564 1644 : R4 Address of PCB (CLEANUP1 entry point only)
0564 1645
```

```

0564 1646 CLEANUP1:
0564 1647 ; Increment enqueue count (if it was charged)
0564 1648
54 00000000'GF D0 0564 1649 MOVL G^SCH$GL CURPCB,R4 ; Get PCB address
05 05 E0 056B 1650 BBS #LKB$V_NOQUOTA,- ; Branch if no quota was charged
08 2A A6 056D 1651 LKB$W_STATUS(R6),CLEANUP2
50 0080 C4 D0 0570 1652 MOVL PCB$JIB(R4),R0 ; Get address of JIB
4C A0 B6 0575 1653 INCW JIB$W_ENQCNT(R0) ; Increment enqueue count
0578 1654
0578 1655 CLEANUP2:
0578 1656 ; Decrement parent LKB's sub LKB reference count.
0578 1657
50 48 A6 D0 0578 1658 MOVL LKB$P_PARENT(R6),R0 ; Get parent LKB address
05 13 057C 1659 BEQL CLEANUP3 ; No parent
4C A0 B7 057E 1660 DECW LKB$W_REFCNT(R0) ; Decrement parent's sub LKB ref. count
15 19 0581 1661 BLSS 90$ ; Ref. count went negative
0583 1662
0583 1663 CLEANUP3:
0583 1664 ; Deallocate RSB. R8 contains RSB address or
0583 1665 ; 0 indicating no RSB to deallocate, R11 contains status.
0583 1666
50 58 D0 0583 1667 MOVL R8,R0 ; Address of RSB
06 13 0586 1668 BEQL CLEANUP4 ; No RSB
00000000'GF 16 0588 1669 JSB G^EXE$DEANONPAGED ; Deallocate it
058E 1670
058E 1671 CLEANUP4:
058E 1672 ; Deallocate LKB. R6 contains LKB address.
058E 1673
50 56 D0 058E 1674 MOVL R6,R0 ; Address of LKB
00000000'GF 16 0591 1675 JSB G^EXE$DEANONPAGED ; Deallocate it
05 05 0597 1676 RSB
0598 1677
0598 1678 90$: BUG CHECK LKBREFNEG,FATAL
059C 1679 .DSABL LSB
```

059C 1681 .SBTTL LCK\$HASH_SEARCH - Hash resource and search hash table
059C 1682
059C 1683 :++
059C 1684 : FUNCTIONAL DESCRIPTION:
059C 1685 :
059C 1686 : This routine hashes the resource name and parent RSB address
059C 1687 : and then searches the hash table looking for a RSB with
059C 1688 : matching:
059C 1689 : o resource names
059C 1690 : o parent RSB addresses
059C 1691 : o access modes
059C 1692 : o name spaces (system or group)
059C 1693 : o group numbers
059C 1694 : Resource name length, access mode, name space and group number
059C 1695 : are all collected into one longword to make the comparison easier.
059C 1696 :
059C 1697 : The entry point LCK\$SRCH_HSHTBL just searches the table using
059C 1698 : a supplied hash value.
059C 1699 :
059C 1700 : CALLING SEQUENCE:
059C 1701 :
059C 1702 : BSBW LCK\$HASH_SEARCH (Hash resource and search table)
059C 1703 : BSBW LCK\$SRCH_HSHTBL (Just search hash table)
059C 1704 : (Note: IPL must be at IPL\$_SYNCH)
059C 1705 :
059C 1706 : INPUT PARAMETERS:
059C 1707 :
059C 1708 : R1 Hash value in low-order 16 bits (LCK\$SRCH_HSHTBL only)
059C 1709 : High order 16 bits must be zero
059C 1710 : R4 Address of parent RSB field (see following ASSUMES)
059C 1711 : R10 Length of resource name + 8
059C 1712 :
059C 1713 : IMPLICIT INPUTS:
059C 1714 :
059C 1715 : This resource's parent's hash value is used to compute this hash
059C 1716 : value.
059C 1717 :
059C 1718 : OUTPUT PARAMETERS:
059C 1719 :
059C 1720 : R0 0 if match found
059C 1721 : 1 if no match found
059C 1722 : R5 Address of matching RSB if a match was found
059C 1723 : R11 Address of last entry in hash chain if no match found
059C 1724 :
059C 1725 : IMPLICIT OUTPUTS:
059C 1726 :
059C 1727 : RSB\$W_HASHVAL is stored with the hash value for this resource
059C 1728 : (LCK\$HASH_SEARCH entry only)
059C 1729 :
059C 1730 : SIDE EFFECTS:
059C 1731 :
059C 1732 : R0 - R3 destroyed
059C 1733 :
059C 1734 : NOTES:
059C 1735 :
059C 1736 : This routine depends on the following fields being adjacent
059C 1737 : in the RSB.


```
0000059C 1738 :--
059C 1739
059C 1740 ASSUME RSB$W_GROUP EQ RSB$L_PARENT+4
059C 1741 ASSUME RSB$B_RMOD EQ RSB$W_GROUP+2
059C 1742 ASSUME RSB$B_RSNLEN EQ RSB$B_RMOD+1
059C 1743 ASSUME RSB$T_RESNAM EQ RSB$B_RSNLEN+1
059C 1744
059C 1745 .IF NDF LOADSW
059C 1746 .PSECT LOCKMGR
059C 1747 .ENDC
059C 1748
059C 1749 LCK$HASH_SEARCH::
059C 1750
059C 1751 ; Zero pad resource name to a longword boundary and get # of
059C 1752 ; longwords in resource name plus parent RSB, group, etc.
059C 1753
53 5A FE 8F 78 059C 1754 ASHL # -2,R10,R3 ; Divide size by 4
50 5A FFFFFFFC 8F CB 05A1 1755 DECL R3 ; Account for parent address
OC 13 05A3 1756 BICL3 #^C<3>,R10,R0 ; Get remainder from 4
51 644A 9E 05AB 1757 BEQL 40$ ; Branch if multiple of 4
81 94 05AD 1758 MOVAB (R4)[R10],R1 ; Get address of end of name
FA 50 04 F2 05B1 1759 30$: CLRB (R1)+ ; Clear to next longword boundary
53 D6 05B3 1760 AOBLS #4,R0,30$
05B7 1761 INCL R3 ; R3 = # of longwords in name
05B9 1762
05B9 1763 ; Fold resource name and auxiliary fields into a single
05B9 1764 ; longword. R3 = number of longwords to combine.
05B9 1765
51 04 A4 DE 05B9 1766 40$: MOVAL 4(R4),R1 ; Pointer to GROUP field
52 D4 05BD 1767 CLRL R2 ; Initialize result reg.
52 52 81 CC 05BF 1768 45$: XORL (R1)+,R2 ; XOR next longword
52 52 09 9C 05C2 1769 ROTL #9,R2,R2 ; and rotate
F6 53 F5 05C6 1770 SOBGTR R3,45$ ; Repeat
50 64 D0 05C9 1771 MOVL (R4),R0 ; Get address of parent RSB
04 13 05CC 1772 BEQL 50$ ; No parent
52 52 44 A0 AC 05CE 1773 XORW RSB$W_HASHVAL(R0),R2 ; XOR parent's hash value
52 A53F19B7 8F C4 05D2 1774 50$: MULL #^XA53F19B7,R2 ; Multiply by unusual number
52 52 10 9C 05D9 1775 ROTL #16,R2,R2 ; Swap words
05DD 1776
05DD 1777 ; Have composite resource name and parent hash value in
05DD 1778 ; the low order word of R2. Store it in the RSB and then
05DD 1779 ; convert it to a hash table entry address (in R5).
05DD 1780
FC A4 52 B0 05DD 1781 MOVW R2,RSB$W_HASHVAL-RSB$L_PARENT(R4) ; Store hash value
51 52 3C 05E1 1782 MOVZWL R2,R1 ; Clear high word and move to R1
05E4 1783
05E4 1784 LCK$SRCH_HSHTBL::
05E4 1785 ; Hash value is in low word of R1. High word must be zero.
05E4 1786
51 51 00000000'GF 78 05E4 1787 ASHL G^LCK$GB_HTBLSHFT,R1,R1 ; Shift to get hash table index
50 00000000'GF D0 05EC 1788 MOVL G^LCK$GL_HASHTBL,R0 ; *** Combine with next instr.
55 6041 DE 05F3 1789 MOVAL (R0)[R1],R5 ; Get address of hash table entry
05F7 1790
05F7 1791 ; R5 = hash table entry address. Search hash table looking for
05F7 1792 ; a resource block with a matching resource name, parent RSB,
05F7 1793 ; access mode, name space (system or group), and group number.
05F7 1794
```

			05F7	1795		ASSUME	RSB\$\$_HSHCHN	EQ	0	
			05F7	1796						
	5B	55	D0	05F7	1797	60\$:	MOVL	R5,R11		; R11 will point to previous entry and
	55	65	D0	05FA	1798		MOVL	(R5),R5		; R5 will point to next RSB in hash chain
		08	13	05FD	1799		BEQL	90\$		; End of chain - resource not found
64	48	A5	SA	29	05FF	1800	CMPC3	R10,RSB\$\$_PARENT(R5),(R4)		; Are the names the same?
		F1	12	0604	1801		BNEQ	60\$		; No, but keep looking
			05	0606	1802		RSB			; Yes, found a match; R0 = 0 from CMPC3
				0607	1803					
	50	01	D0	0607	1804	90\$:	MOVL	#1,R0		; No match found
			05	060A	1805		RSB			

060B 1807 .SBTTL LCK\$GRANT_LOCK - Grant a lock request
060B 1808
060B 1809 :++
060B 1810 : FUNCTIONAL DESCRIPTION:
060B 1811 :
060B 1812 : This routine performs the actual granting of a lock. This includes:
060B 1813 : o Computing the new group grant mode
060B 1814 : o Inserting the LKB on the granted queue
060B 1815 : o Setting the event flag (if required)
060B 1816 : o Delivering the completion AST (if required)
060B 1817 : o Delivering the blocking AST (if required)
060B 1818 :
060B 1819 : This routine gets called for both synchronous grants (in the context
060B 1820 : of the process requesting a lock) and asynchronous grants (in the
060B 1821 : context of another process that has just performed a dequeue or
060B 1822 : a conversion to a lower lock mode).
060B 1823 :
060B 1824 : The alternate entry point LCK\$GRANT_LOCK_ALT is used when the caller
060B 1825 : knows that there are no granted locks (or waiting conversions) for
060B 1826 : this resource.
060B 1827 :
060B 1828 : The entry point LCK\$REGRANTLOCK is used to regrant an attempted
060B 1829 : conversion that was put on the conversion queue and then taken
060B 1830 : off due to deadlock. Note that in this case the LKBSM_ASYNC
060B 1831 : bit must be set (even though we might be completing the request
060B 1832 : synchronously) in order to execute the code that moves the lock from
060B 1833 : the tail to the head of the PCB lock queue.
060B 1834 :
060B 1835 : The QUEUE_AST entry point is used to just queue an AST when
060B 1836 : dequeuing an ungranted lock (abort or deadlock).
060B 1837 :
060B 1838 : CALLING SEQUENCE:
060B 1839 :
060B 1840 : BSBW LCK\$GRANT_LOCK
060B 1841 : Note: IPL must be at IPL\$SYNCH
060B 1842 :
060B 1843 : INPUT PARAMETERS:
060B 1844 :
060B 1845 : All entry points:
060B 1846 :
060B 1847 : R1 Lock mode to grant
060B 1848 : R6 Address of LKB
060B 1849 : R8 Address of RSB
060B 1850 :
060B 1851 : LCK\$GRANT_LOCK:
060B 1852 :
060B 1853 : R5 Existing group grant mode
060B 1854 :
060B 1855 : IMPLICIT INPUTS:
060B 1856 :
060B 1857 : LCK\$REGRANTLOCK and QUEUE_AST:
060B 1858 :
060B 1859 : LKBSL_LKST1 contains final completion status
060B 1860 : Also the LKBSM_ASYNC bit must be set
060B 1861 :
060B 1862 : OUTPUT PARAMETERS:
060B 1863 :

```
060B 1864 : LCK$GRANT_LOCK and LCK$GRANT_LOCK_ALT:
060B 1865 :
060B 1866 : R0 Request completion code
060B 1867 : R5 New group grant mode
060B 1868 :
060B 1869 : IMPLICIT OUTPUTS:
060B 1870 :
060B 1871 : LCK$GRANT_LOCK and LCK$GRANT_LOCK_ALT:
060B 1872 :
060B 1873 : SS$_NORMAL is stored in the 1st longword of the LKSB
060B 1874 :
060B 1875 : COMPLETION CODES:
060B 1876 :
060B 1877 : SS$_SYNCH Synchronous completion
060B 1878 : SS$_NORMAL Normal completion
060B 1879 :
060B 1880 : SIDE EFFECTS:
060B 1881 :
060B 1882 : R1 - R4 are destroyed
060B 1883 : --
060B 1884 :
060B 1885 : .IF NDF LOADSW
0000 060B 1886 : .PSECT LOCKMGR
060B 1887 : .ENDC
060B 1888 :
060B 1889 : .ENABLE LSB
060B 1890 :
060B 1891 : LCK$GRANT_LOCK::
55 51 91 060B 1892 : CMPB R1,R5 ; Should there be a new group grant mode?
060B 1893 : BLEQU 10$ ; No
0610 1894 :
0610 1895 : LCK$GRANT_LOCK_ALT::
0C A8 51 90 0610 1896 : MOVB R1,RSB$_GGMODE(R8) ; Yes, store group grant mode
55 51 D0 0614 1897 : MOVL R1,R5 ; and in R5
0D A8 51 90 0617 1898 : MOVB R1,RSB$_CGMODE(R8) ; Store conversion grant mode
061B 1899 :
35 A6 51 90 061B 1900 10$: MOVB R1,LKB$_GRMODE(R6) ; Store granted lock mode
061F 1901 : MOVZWL S^#SS$_NORMAL,- ; Store success in LKSB
2C A6 3C 0621 1902 : LKB$_LKST1(R6)
0623 1903 : LCK$REGRANTLOCK::
0623 1904 :
0623 1905 : ; Determine if this request wants a blocking AST. If yes,
0623 1906 : ; then increment blocking AST count in RSB and determine if we
0623 1907 : ; are blocking anyone.
0623 1908 :
0623 1909 : ASSUME RSB$_WTQFL EQ RSB$_CVTQFL+8
0623 1910 :
20 A6 D5 0623 1911 : TSTL LKB$_BLKASTADR(R6) ; Blocking AST address specified?
35 13 0626 1912 : BEQL 30$ ; No
42 A8 B6 0628 1913 : INCW RSB$_BLKASTCNT(R8) ; Incr. blocking AST count
52 18 A8 DE 062B 1914 : MOVAL RSB$_CVTQFL(R8),R2 ; Get address of conversion queue
50 02 D0 062F 1915 : MOVL #2,R0 ; Do this loop twice
0632 1916 :
53 52 D0 0632 1917 15$: MOVL R2,R3 ; Save address of queue header
53 63 D0 0635 1918 20$: MOVL (R3),R3 ; Get address of next element
53 52 D1 0638 1919 : CMPL R2,R3 ; Is it the header?
1A 13 063B 1920 : BEQL 25$ ; Yes, done with this queue
```

```
ED F9B9 54 FC A3 9A 063D 1921 MOVZBL LKBSB_RMODE-LKBSL_SQFL(R3),R4 ; No, get requested lock mode
          CF44 51 E0 0641 1922 BBS R1,LCK$COMPAT_TBL[R4],20$ ; Br. if compatible
          2A A6 0A A8 0648 1923 BISW #LKBSM_BLKASTQED!- ; Set blocking AST queued and
          10 88 064C 1924 LKBSM_DBLKAST,LKBSW_STATUS(R6) ; deliver blocking AST status
          0B A6 064E 1925 BISB #LKBSM_PKAST,- ; Set piggyback kernel AST bit
          20 A6 D0 0650 1926 MOVL LKBSB_RMOD(R6)
          10 A6 0653 1927 LKBSL_BLKASTADR(R6),- ; Store address of blocking AST routine
          06 11 0655 1928 LKBSL_AST(R6)
          52 08 C0 0657 1929 BRB 30$ ; Search no more
          D5 50 F5 065A 1930 25$: ADDL #8,R2 ; Advance to wait queue
          065D 1931 SOBGTR R0,15$ ; Repeat
          065D 1932
          065D 1933 LCK$GRANT_REM::
          065D 1934 30$: ; Set state to granted and insert on granted queue.
          065D 1935 ; Determine if a completion AST should be queued.
          065D 1936
          38 A6 0E 065D 1937 INSQUE LKBSL_SQFL(R6),- ; Insert lock on granted queue
          10 A8 0660 1938 RSB$-GRQFL(R8)
          01 90 0662 1939 MOVB #LKBSR_GRANTED,- ; Indicate lock is on granted queue
          36 A6 0664 1940 LKBSB_STATE(R6)
          0666 1941
          50 0689 8F 3C 0666 1942 MOVZWL #SS$ SYNCH,R0 ; Assume synchronous completion
          10 B3 066B 1943 BITW #LKBSM_MSTCPY,- ; Branch if this a
          2A A6 066D 1944 LKBSW_STATUS(R6) ; master copy
          13 12 066F 1945 BNEQ 35$
          04 B3 0671 1946 BITW #LKBSM_ASYNC,- ; Branch if this request is being
          2A A6 0673 1947 LKBSW_STATUS(R6) ; completed asynchronously
          1F 12 0675 1948 BNEQ 40$
          08 B3 0677 1949 BITW #LCK$M_SYNCSTS,- ; Branch if SYNCSTS bit is set
          28 A6 0679 1950 LKBSW_FLAGS(R6)
          66 12 067B 1951 BNEQ 80$
          1C A6 D5 067D 1952
          4D 13 0680 1953 TSTL LKBSL_CPLASTADR(R6) ; Did caller specify a completion AST?
          3E 11 0682 1954 BEQL 70$ ; No, just set event flag
          0684 1955 BRB 50$ ; Yes, set appropriate bits
          0684 1956
          0684 1957 35$: ; Come here if the lock is a master copy
          0684 1958
          04 B3 0684 1959 BITW #LKBSM_ASYNC,- ; Branch if this request is being
          2A A6 0686 1960 LKBSW_STATUS(R6) ; completed synchronously
          0B 13 0688 1961 BEQL 37$
          55 DD 068A 1962 PUSHL R5 ; Save group grant mode
          00000000 GF 16 068C 1963 JSB G^LCK$SND_GRANTED ; Send a granted message
          55 8ED0 0692 1964 POPL R5 ; Restore it
          05 0695 1965 37$: RSB
          0696 1966
          0696 1967 40$: ; The request is being completed asynchronously so it is necessary
          0696 1968 ; to remove the lock from its current position in the PCB queue
          0696 1969 ; (around the tail) and reinsert it at the head of the PCB queue.
          0696 1970
          54 0C A6 3C 0696 1971 MOVZWL LKBSL_PID(R6),R4 ; Get process index
          50 00000000 GF D0 069A 1972 MOVL G^SCH$GL_PCBVEC,R0 ; *** Combine this and next inst. when
          06A1 1973 ; PIC code is no longer needed ***
          54 6044 D0 06A1 1974 MOVL (R0)[R4],R4 ; Convert to PCB address
          50 40 A6 0F 06A5 1975 REMQUE LKBSL_OWNOFL(R6),R0 ; Remove lock from PCB lock queue
          0104 C4 60 0E 06A9 1976 INSQUE (R0),PCBSL_LOCKQFL(R4) ; Insert at head of lock queue
          06AE 1977
```

```
06AE 1978 QUEUE_AST:
06AE 1979 ; Request is being completed asynchronously so a kernel AST is
06AE 1980 ; required to store status. This is also used as an entry point
06AE 1981 ; to queue an AST when dequeuing an ungranted lock. (e.g. abort
06AE 1982 ; or deadlock). Status should already be in LKB$AST. Note:
06AE 1983 ; The LKB$M_NODELETE bit may have been cleared before calling QUEUE_AST.
06AE 1984
03 2A 06 E5 06AE 1985 BBCC #LKB$V_TIMEOUTQ,- ; Remove lock from timeout queue
50 66 0F 06B0 1986 LKB$W_STATUS(R6),45$ ; if it was on it
1C A6 D5 06B3 1987 REMQUE LKB$ASTQFL(R6),R0
07 12 06B6 1988 45$: TSTL LKB$CPLASTADR(R6) ; Did caller specify a completion AST?
80 8F 88 06B9 1989 BNEQ 50$ ; Yes
0B A6 06BE 1990 BISB #LKB$M_KAST,- ; No, set special kernel AST bit
09 11 06C0 1991 LKB$B_RMOD(R6)
06C2 1992 BRB 60$ ; and deliver completion AST anyway
06C2 1993 50$: ; A completion AST was requested by the caller.
06C2 1994
0B 10 88 06C2 1996 BISB #LKB$M_PKAST,- ; Set piggyback kernel AST flag
1C A6 D0 06C4 1997 LKB$B_RMOD(R6)
10 A6 06C6 1998 MOVL LKB$CPLASTADR(R6),- ; Store completion AST address
01 A8 06C9 1999 LKB$AST(R6)
2A A6 06CB 2000 60$: BISW #LKB$M_DCPLAST,- ; Set deliver completion AST flag
06CD 2001 LKB$W_STATUS(R6)
06CF 2002 70$: ; Set the event flag
06CF 2003
51 0C A6 D0 06CF 2004 MOVL LKB$PID(R6),R1 ; Get PID
52 02 9A 06D3 2006 MOVZBL #PRI$-RESAVL,R2 ; Priority increment class
53 37 A6 9A 06D6 2007 MOVZBL LKB$B_EFN(R6),R3 ; Event flag number
00000000'GF 16 06DA 2008 JSB G^SCH$POSTEF ; Post event flag
50 01 3C 06E0 2009 MOVZWL S^SS$NORMAL,R0 ; Return success
06E3 2010 80$: ; Queue AST if either completion or blocking AST flags are set.
06E3 2011
2A A6 03 B3 06E3 2013 BITW #LKB$M_DCPLAST!,- ; Is either flag set?
06E7 2014 LKB$M_DBLKAST,LKB$W_STATUS(R6)
06E7 2015 BEQL 90$ ; No
55 21 BB 06E9 2016 PUSHR #^M<R0,R5> ; Save R0 and R5
0C A6 D5 06EB 2017 MOVL R6,R5 ; R5 points to ACB
11 13 06EE 2018 TSTL LKB$PID(R6) ; Is this a system owned lock?
0A AF 9E 06F1 2019 BEQL 95$ ; Yes
18 A6 06F3 2020 MOVAB B^LOCK_KAST,- ; Store address of kernel AST routine
52 02 9A 06F6 2021 LKB$AST(R6)
00000000'GF 16 06F8 2022 MOVZBL #PRI$-RESAVL,R2 ; Priority increment class
21 BA 0701 2023 JSB G^SCH$QAST ; Yes, queue AST
05 0703 2024 POPR #^M<R0,R5> ; Restore regs
0704 2025 90$: RSB
0704 2026 95$: ; Handle system owned locks. Since we can't be delivering a
0704 2027 ; completion AST, we must have to deliver a blocking AST
0704 2028 ; (actually a blocking subroutine call).
0704 2029
0198 30 0704 2031 BSBW CALL_BLK_SUBR ; Call blocking subroutine
21 BA 0707 2032 POPR #^M<R0,R5> ; Restore regs
05 0709 2033 RSB
070A 2034
```

SYSENQDEQ
V04-000

- ENQUEUE/DEQUEUE SYSTEM SERVICES^{N 1}
LCK\$GRANT_LOCK - Grant a lock request
070A 2035 .DSABL LSB

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 44
(14)

SY
VC

```
070A 2037 .SBTTL LOCK_KAST - Kernel AST routine
070A 2038
070A 2039 :++
070A 2040 : FUNCTIONAL DESCRIPTION:
070A 2041 :
070A 2042 : This routine is the special kernel AST routine. It is called
070A 2043 : whenever completion or blocking ASTs are delivered.
070A 2044 : If a completion AST is being delivered and the request is being
070A 2045 : completed asynchronously, then the lock status block and
070A 2046 : optionally the value block are copied to the caller's LKSB.
070A 2047 : Then (synch. or async.) we see if a blocking AST is required.
070A 2048 : If it is then we requeue the LKB to deliver the blocking AST.
070A 2049 : If a blocking AST is being delivered then we just have to clear
070A 2050 : the bit that tells us to deliver a blocking AST. Finally, if the
070A 2051 : LKB is not requeued for an AST we check to see if it should be
070A 2052 : deleted and if so, delete it.
070A 2053 :
070A 2054 : CALLING SEQUENCE:
070A 2055 :
070A 2056 : JSB LOCK_KAST (actually called by SCH$ASTDEL)
070A 2057 : Note: This routine may be called as either a special kernel
070A 2058 : AST routine or piggyback kernel AST routine or both.
070A 2059 :
070A 2060 : INPUT PARAMETERS:
070A 2061 :
070A 2062 : R4 Address of PCB
070A 2063 : R5 Address of LKB
070A 2064 : Note: Only offsets up to LKBSK_ACBLEN may be used
070A 2065 :
070A 2066 : OUTPUT PARAMETERS:
070A 2067 :
070A 2068 : None
070A 2069 :
070A 2070 : SIDE EFFECTS:
070A 2071 :
070A 2072 : ???
070A 2073 :--
070A 2074 :
070A 2075 : .IF NDF LOADSW
0000070A 2076 : .PSECT LOCKMGR
070A 2077 : .ENDC
070A 2078 :
070A 2079 LOCK_KAST:
179 2A 00 E1 070A 2080 BBC #LKBSV_DCPLAST, - ; Branch if no completion AST to deliver
070C 2081 LKBSW_STATUS(R5),40$
139 2A 02 E1 070F 2082 BBC #LKBSV_ASYNC, - ; Branch if synchronous completion
0711 2083 LKBSW_STATUS(R5),20$
0714 2084
0714 2085 ; Store status and value block, if requested. If the value block
0714 2086 ; is invalid then the status in LKBSL_LKST1 is changed. Note that
0714 2087 ; this only happens if the status code was a success code.
0714 2088
151 24 A5 D0 0714 2089 MOVL LKBSL_LKSB(R5),R1 ; Get address of LKSB
0718 2090 BBC #LKBSV_VALBLK, - ; Branch if no value block
126 28 A5 071A 2091 LKBSW_FLAGS(R5),10$
122 2C A5 E9 071D 2092 BLBC LKBSL_LKST1(R5),10$ ; Only store value block if request
0721 2093 ; was successful
```



```

50 50 A5 DO 0721 2094 IFNOWRT #24,(R1),20$ ; Branch if LKSB is not writeable
      02 B3 0727 2095 MOVL LKB$L_RSB(R5),R0 ; R0 = address of RSB
      OE A0 072B 2096 BITW #RSB$M_VALINVL,- ; Is value block valid?
      06 13 072D 2097 RSB$W_STATUS(R0)
      09F0 8F B0 072F 2098 BEQL 5$ ; Yes
      2C A5 0731 2099 MOVW #SS$_VALNOTVALID,- ; No, change completion status
      08 A1 28 A0 7D 0735 2100 LKB$C_LKST1(R5)
      10 A1 30 A0 7D 0737 2101 5$: MOVQ RSB$Q_VALBLK(R0),8(R1) ; Store value block - Note: the fact
      073C 2102 MOVQ RSB$Q_VALBLK+8(R0),16(R1) ; that we always store the value block
      0741 2103 ; is based on the assumption that $ENQs
      0741 2104 ; that fetch it are always synchronous.
      06 11 0741 2105 BRB 15$
      61 2C A5 DO 0743 2106 10$: IFNOWRT #8,(R1),20$ ; Branch if LKSB is not writeable
      0749 2107 15$: MOVL LKB$L_LKST1(R5),(R1) ; Store status
      074D 2108 ;
      074D 2109 20$: ; Requeue LKB if we have to deliver a blocking AST. Also, convert
      074D 2110 ; lock back to system owned, if necessary.
      074D 2111
      074D 2112 SETIPL #IPL$ SYNCH ; Raise IPL
      01 AA 0750 2113 BICW #LKB$M_DCPLAST,- ; Clear deliver completion AST bit
      2A A5 0752 2114 LKB$W_STATUS(R5)
      08 E0 0754 2115 BBS #LKB$V_CVTOSYS,- ; Branch if this lock should be con-
      15 2A A5 0756 2116 LKB$W_STATUS(R5),30$ ; verted back to system owned
      01 E1 0759 2117 BBC #LKB$V_DBLKAST,- ; Branch if no blocking AST to deliver
      31 2A A5 075B 2118 LKB$W_STATUS(R5),60$
      20 A5 DO 075E 2119 MOVL LKB$L_BLKASTADR(R5),- ; Store blocking AST address
      10 A5 0761 2120 LKB$L_AST(R5)
      52 02 9A 0763 2121 MOVZBL #PRI$-RESAVL,R2 ; Priority increment class
      00000000 GF 16 0766 2122 JSB G^SCH$QAST ; Queue AST
      3C 11 076C 2123 BRB 80$
      076E 2124
      076E 2125 30$: ; Lock needs to be converted back to system owned. Restore old AST
      076E 2126 ; parameter and convert back to system owned. Call blocking
      076E 2127 ; AST subroutine instead of queueing an AST, if necessary.
      076E 2128 ; Note that this path should NEVER be taken if the NODELETE bit is
      076E 2129 ; clear (i.e. places that clear the NODELETE bit should also clear
      076E 2130 ; CVTOSYS). There are two reasons for this. First, if NODELETE
      076E 2131 ; is clear, then the LKB may already be removed from the PCB lock
      076E 2132 ; queue and CVT TO SYS would do a double REMQUE. Secondly, the field
      076E 2133 ; LKB$L_OLDASTPRM is only part of the full LKB; it's not part of
      076E 2134 ; the ACB portion of the LKB. Therefore, this code path can only be
      076E 2135 ; used if we are dealing with the full LKB (see routine FREE LKB).
      076E 2136 ; It is for these reasons that this code path branches to 80$
      076E 2137 ; instead of 60$ when it's finished.
      076E 2138
      58 A5 DO 076E 2139 MOVL LKB$L_OLDASTPRM(R5),- ; Restore old AST parameter
      14 A5 0771 2140 LKB$L_ASTPRM(R5)
      56 56 DD 0773 2141 PUSHL R6 ; Save R6
      56 55 DO 0775 2142 MOVL R5,R6 ; Move LKB address
      FA60 30 0778 2143 BSBW CVT_TO_SYS_INT ; Convert to system owned
      56 8ED0 077B 2144 POPL R6 ; Restore R6
      01 E1 077E 2145 BBC #LKB$V_DBLKAST,- ; Branch if no blocking AST to deliver
      27 2A A5 0780 2146 LKB$W_STATUS(R5),80$
      0119 30 0783 2147 BSBW CALL_BLK_SUBR ; Call blocking subroutine
      22 11 0786 2148 BRB 80$
      0788 2149
      0788 2150 40$: ; We must be delivering a blocking AST
```

```
0788 2151
0788 2152
2A 02 AA 078B 2153 SETIPL #IPL$ SYNCH ; Raise IPL
078D 2154 BICW #LKB$M_DBLKAST, - ; Clear deliver blocking AST bit
078F 2155 LKB$W_STATUS(R5)
078F 2156 60$: ; Delete LKB if no longer needed. Increment enqueue quota unless
078F 2157 ; the NOQUOTA bit in the LKB is set. In this case,
078F 2158 ; AST quota has already been accounted for in the AST delivery code.
078F 2159
16 0B 05 E0 078F 2160 BBS #LKB$V_NODELETE, - ; Branch if not deleting
0791 2161 LKB$B_RMOD(R5), 80$
08 2A 05 E0 0794 2162 BBS #LKB$V_NOQUOTA, - ; Branch if accounting taken care of
0796 2163 LKB$W_STATUS(R5), 70$
50 0080 C4 D0 0799 2164 MOVL PCBSL_JIB(P4), R0 ; Get JIB address
4C A0 B6 079E 2165 INCW JIB$W_ENQCNT(R0) ; Add 1 to enqueue count
50 55 D0 07A1 2166 70$: MOVL R5, R0
00000000 GF 16 07A4 2167 JSB G^EXE$DEANONPAGED ; Deallocate it
07AA 2168 80$: SETIPL #IPL$_ASTDEL ; Lower !PL
07AD 2169 RSB
07AE 2170
07AE 2171
07AE 2172 ;
07AE 2173 ; Synchronization notes:
07AE 2174 ;
07AE 2175 ; 1) The clearing of the DCPLAST bit and the testing of the
07AE 2176 ; DBLKAST bit (after label 20$) must be done at IPL$ SYNCH
07AE 2177 ; in order to synchronize correctly with the queueing of a
07AE 2178 ; blocking AST by routine LCK$QUEUE_BLOCKAST. Otherwise, the AST
07AE 2179 ; could be queued twice.
07AE 2180 ;
07AE 2181 ; 2) Running the rest of the routine at IPL$ SYNCH (labels 40$ and
07AE 2182 ; 80$) is just done for safety's sake. At this writing, it
07AE 2183 ; is not believed to be necessary.
07AE 2184 ;
```

```
07AE 2186      .SBTTL LCK$QUEUECVT - Insert a lock on conversion queue
07AE 2187      .SBTTL LCK$QUEUEWAIT - Insert a lock on wait queue
07AE 2188
07AE 2189      ;++
07AE 2190      : FUNCTIONAL DESCRIPTION:
07AE 2191      :
07AE 2192      : These routines handle lock requests when they cannot be granted
07AE 2193      : immediately. LCK$QUEUECVT handles conversion requests and
07AE 2194      : LCK$QUEUEWAIT handles new lock requests. These routines also
07AE 2195      : queue the lock onto a timeout queue if deadlock checking is enabled.
07AE 2196
07AE 2197      : CALLING SEQUENCE:
07AE 2198      :
07AE 2199      : BSBW LCK$QUEUECVT (or LCK$QUEUEWAIT)
07AE 2200      : (Note: IPL must be at IPL$_SYNCH)
07AE 2201
07AE 2202      : INPUT PARAMETERS:
07AE 2203      :
07AE 2204      : R4      Address of PCB
07AE 2205      : R5      Group grant mode without this lock (LCK$QUEUECVT only)
07AE 2206      : R6      Address of LKB
07AE 2207      : R8      Address of RSB
07AE 2208      : R9      Flags
07AE 2209
07AE 2210      : OUTPUT PARAMETERS:
07AE 2211      :
07AE 2212      : None
07AE 2213
07AE 2214      : SIDE EFFECTS:
07AE 2215      :
07AE 2216      : All registers except R6 may be clobbered.
07AE 2217
07AE 2218      :--
07AE 2219
07AE 2220      .IF NDF LOADSW
07AE 2221      .PSECT LOCKMGR
07AE 2222      .ENDC
07AE 2223
07AE 2224      .ENABL LSB
07AE 2225
07AE 2226      ASSUME LKBSK_CONVERT EQ 0
07AE 2227
07AE 2228 LCK$QUEUECVT::
07AE 2229     INSQUE LKBSL_SQFL(R6),-      : Insert at end of conversion queue
07AE 2230     @RSBSL_CVTQBL(R8)
07AE 2231     BNEQ 10$      : Branch if not first in queue
07AE 2232     MOVB R5,RSBSB_CGMODE(R8)      : Store conversion grant mode
07AE 2233     CLRB LKBSB_STATE(R6)        : Set state = conversion
07AE 2234     BITW #LKBSM_MSTCPY,-        : Is this is a master copy?
07AE 2235     LKBSW_STATUS(R6)
07AE 2236     BNEQ 15$      : Yes
07AE 2237     REMQUE LKBSL_OWNOFL(R6),R0    : Remove from PCB lock queue
07AE 2238     BRB QUEUE_COMMON            : Join common code
07AE 2239
07AE 2240 LCK$QUEUEWAIT::
07AE 2241     INSQUE LKBSL_SQFL(R6),-      : Insert request at end of wait queue
07AE 2242     @RSBSL_WTQBL(R8)
```

000007AE 2221 .PSECT LOCKMGR

07AE 2222 .ENDC

07AE 2223

07AE 2224 .ENABL LSB

07AE 2225

07AE 2226 ASSUME LKBSK_CONVERT EQ 0

07AE 2227

07AE 2228 LCK\$QUEUECVT::

07AE 2229 INSQUE LKBSL_SQFL(R6),-

07AE 2230 @RSBSL_CVTQBL(R8)

07AE 2231 BNEQ 10\$

07AE 2232 MOVB R5,RSBSB_CGMODE(R8)

07AE 2233 CLRB LKBSB_STATE(R6)

07AE 2234 BITW #LKBSM_MSTCPY,-

07AE 2235 LKBSW_STATUS(R6)

07AE 2236 BNEQ 15\$

07AE 2237 REMQUE LKBSL_OWNOFL(R6),R0

07AE 2238 BRB QUEUE_COMMON

07AE 2239

07AE 2240 LCK\$QUEUEWAIT::

07AE 2241 INSQUE LKBSL_SQFL(R6),-

07AE 2242 @RSBSL_WTQBL(R8)

38 A6 0E 07AE 2229

1C B8 07AE 2230

04 12 07AE 2231

OD A8 55 90 07AE 2232

36 A6 94 07AE 2233

10 B3 07AE 2234

2A A6 07AE 2235

26 12 07AE 2236

50 40 A6 0F 07AE 2237

10 11 07AE 2238

07AE 2239

07AE 2240

38 A6 0E 07AE 2241

24 B8 07AE 2242

```
FF 8F 90 07CD 2243 LCK$QUEUE REM:
36 A6 07CD 2244      MOVB   #LKB$K_WAITING,-      ; Set state = waiting
      10 B3 07D0 2245      LKB$B_STATE(R6)
2A A6 07D2 2246      BITW   #LKB$M_MSTCPY,-      ; Is this is a master copy?
      10 07D4 2247      LKB$W_STATUS(R6)
      12 77D6 2248      BNEQ   15$              ; Yes
      07D8 2249
      07D8 2250 QUEUE_COMMON:
      07D8 2251      ; Insert lock on process lock queue and clear event flag.
      07D8 2252
      07D8 2253      INSQUE  LKB$L_OWNQFL(R6),-      ; Insert at end of PCB lock queue
      07D8 2254      @PCB$C_LOCKQBL(R4)
      07D8 2255      MOVZBL  LKB$B_EFN(R6),R3      ; Get event flag number
53 37 A6 9A 07DE 2256      JSB   G^SCH$CLREFR      ; Clear the event flag
00000000'GF 07E2 2257
      07E8 2258
      07E8 2259 15$:      ; Set ASYNC bit and queue blocking ASTs
      07E8 2260
      07E8 2261      .IF NE  CAS MEASURE
00000000'GF 06 07E8 2262      INCL  G^PM$SGL_ENQWAIT
      07EE 2263      .ENDC
      07EE 2264
      07EE 2265      BISW   #LKB$M_ASYNC,-      ; Set bit to indicate this request
2A A6 07F0 2266      LKB$W_STATUS(R6)      ; will be satisfied asynchronously
2C A6 07F2 2267      CLRL   LKB$L_LKST1(R6)      ; Clear 1st longword of LKSB
42 A8 B5 07F5 2268      TSTW   RSB$W_BLKASTCNT(R8) ; See if anyone wants a blocking AST
      02 13 07F8 2269      BEQL   20$              ; No
      31 10 07FA 2270      BSBB   LCK$QUEUE_BLOCKAST ; Yes, queue blocking ASTs
      07FC 2271
      07FC 2272 20$:      ; Insert this lock on timeout queue if deadlock checking is enabled
      07FC 2273      ; and this lock is mastered on this system and LCK$M_NODLCKWT is
      07FC 2274      ; not set.
      07FC 2275
      07FC 2276      ASSUME  LKB$L_DUETIME EQ LKB$L_KAST ; Note that these fields overlap
      07FC 2277
      07FC 2278      TSTL   RSB$L_CSID(R8)      ; Don't insert process copy locks
38 A8 07FF 2279      BNEQ   30$
      28 12 0801 2280      BBS   #LCK$V_NODLCKWT,-      ; Branch if no deadlock wait is set
      09 E0 0803 2281      LKB$W_FLAGS(R6),30$
26 28 A6 0806 2282      MOVL   G^LCK$GL_WAITTIME,R0 ; Get lock wait time
50 00000000'GF 1D 13 080D 2283      BEQL   30$      ; Deadlock checking is disabled
00000000'GF 50 C1 080F 2284      ADDL3  R0,G^EXE$GL_ABSTIM,- ; Add wait time to current time to
      18 A6 0816 2285      LKB$L_DUETIME(R6)      ; get duetime
      4E A6 94 0818 2286      CLRB   LKB$B_TSLT(R6) ; Init. timestamp lifetime
0040 8F A8 081B 2287      BISW   #LKB$M_TIMEOUTQ,- ; Set timeout queue bit
2A A6 081F 2288      LKB$W_STATUS(R6)
50 00000000'GF DE 0821 2289      MOVAL  G^LCK$GL_TIMEOUTQ,R0
      66 0E 0828 2290      INSQUE  LKB$L_ASTQFL(R6),- ; Insert lock on end of timeout queue
      04 B0 082A 2291      @4(R0)
      05 082C 2292 30$:      RSB
      082D 2293
      082D 2294      .DSABL  LSB
```

```
082D 2296 .SBTTL LCK$QUEUE_BLOCKAST - Queue blocking ASTs
082D 2297
082D 2298 :++
082D 2299 : FUNCTIONAL DESCRIPTION:
082D 2300 :
082D 2301 : This routine queues a blocking AST to all locks that meet the
082D 2302 : following conditions:
082D 2303 : o Are on the granted queue
082D 2304 : o Have requested a blocking AST
082D 2305 : o Have not already received a blocking AST
082D 2306 : o Whose granted lock mode is incompatible with
082D 2307 : the requested lock mode of the lock being placed in
082D 2308 : the conversion or wait queue
082D 2309 : o Whose lock state is GRANTED (eliminates locks in a SCS
082D 2310 : conversion wait state)
082D 2311 :
082D 2312 : This routine assumes that the caller has already determined
082D 2313 : that RSB$W_BLKASTCNT is non-zero, indicating that there is at least
082D 2314 : one lock requesting a blocking AST.
082D 2315 :
082D 2316 : CALLING SEQUENCE:
082D 2317 :
082D 2318 : BSBW LCK$QUEUE_BLOCKAST
082D 2319 : (Note: IPL must be at IPL$_SYNCH)
082D 2320 :
082D 2321 : INPUTS:
082D 2322 :
082D 2323 : R6 Address of LKB (being placed on conversion or wait queue)
082D 2324 : R8 Address of RSB
082D 2325 :
082D 2326 : OUTPUTS:
082D 2327 :
082D 2328 : None
082D 2329 :
082D 2330 : IMPLICIT OUTPUTS:
082D 2331 :
082D 2332 : Possibly, a number of blocking ASTs are queued
082D 2333 :
082D 2334 : COMPLETION CODES:
082D 2335 :
082D 2336 : None
082D 2337 :
082D 2338 : SIDE EFFECTS:
082D 2339 :
082D 2340 : R0 - R5, R7, R10, and R11 are destroyed
082D 2341 : --
082D 2342 :
082D 2343 : .IF NDF LOADSW
0000082D 2344 : .PSECT LOCKMGR
082D 2345 : .ENDC
082D 2346 :
082D 2347 : LCK$QUEUE_BLOCKAST::
082D 2348 : MOVZBL LKB$B_RQMODE(R6),R10 ; Get req. lock mode of blocked lock
082D 2349 : MOVAL RSB$B_GRQFL(R8),R11 ; Get address of granted queue
082D 2350 : MOVL R11,R7 ; Save address
082D 2351 : 10$: MOVL (R7),R7 ; Get address of next element in queue
082D 2352 : CMPL R7,R11 ; Reached the end yet?
```

```

55 57 37 13 083E 2353 BEQL 90$ ; Yes
20 A5 D5 0840 2354 SUBL3 #LKB$S_SQFL,R7,R5 ; No, position to start of LKB
EF 13 0844 2355 TSTL LKB$S_BLKASTADR(R5) ; Blocking AST address specified?
50 35 A5 9A 0847 2356 BEQL 10$ ; No
E4 F7AD CF4A 50 E0 0849 2357 MOVZBL LKB$B_GMODE(R5),R0 ; Get granted lock mode
03 E0 084D 2358 BBS R0,LCK$COMPAT_TBL[R10],10$ ; Branch if compatible
DF 2A A5 0854 2359 BBS #LKB$V_BLKASTQED,- ; Branch if blocking ast already
36 A5 91 0856 2360 LKB$W_STATUS(R5),10$ ; queued
01 085C 2361 CMPB LKB$B_STATE(R5),- ; Is lock granted?
D9 12 085D 2362 #LKB$R_GRANTED
10 B3 085F 2363 BNEQ 10$ ; No
2A A5 0861 2364 BITW #LKB$M_MSTCPY,- ; Is this a master copy?
0A 12 0863 2365 LKB$W_STATUS(R5)
0865 2366 BNEQ 80$ ; Yes - send a message to other system
00000000'GF 00000002 0865 2367 .IF NE CAS MEASURE
0865 2368 INCL G^PM$SGL_BLK_LOC
086B 2369 .ENDC
086B 2370
08 10 086B 2371 BSBB LCK$QUEUE_BLKAST ; No, actually queue an AST
C9 11 086D 2372 BRB 10$ ; Repeat for remaining locks
086F 2373
00000000'GF 16 086F 2374 80$: JSB G^LCK$SND_BLKING ; Send a blocking message
C1 11 0875 2375 BRB 10$ ; Repeat for remaining locks
0877 2376
05 0877 2377 90$: RSB
0878 2378
0878 2379
0878 2380 ;++
0878 2381 ; Subroutine to actually queue the blocking AST.
0878 2382 ;
0878 2383 ; Input:
0878 2384 ; R5 Address of LKB
0878 2385 ; Output:
0878 2386 ; R0 - R5 not preserved
0878 2387 ;--
0878 2388
0878 2389 .ENABL LSB
0878 2390
0C A5 D5 0878 2391 LCK$QUEUE_BLKAST::
22 13 0878 2392 TSTL LKB$S_PID(R5) ; Is this lock system owned?
087B 2393 BEQL CALL_BLK_SUBR ; Yes, call blocking subroutine
087D 2394
087D 2395 QUEUE_BLKAST:
087D 2396 ; Deliver a blocking AST to the process owning this lock (LKB in R5)
087D 2397
2A A5 0A A8 087D 2398 BISW #LKB$M_BLKASTQED!- ; Set blocking AST queued and
0881 2399 LKB$M_DBLKAST,LKB$W_STATUS(R5) ; deliver blocking AST status
10 88 0881 2400 BISB #LKB$M_PKAST,- ; Set piggyback kernel AST bit
08 0883 2401 LKB$B_RMOD(R5)
00 E0 0885 2402 BBS #LKB$V_DCPLAST,- ; Branch if the LKB is already queued
14 2A A5 0887 2403 LKB$W_STATUS(R5),70$ ; NOTE: This test is based on the
088A 2404 ; assumption that the LKB$M_DBLKAST bit
088A 2405 ; cannot be set since the
088A 2406 ; LKB$M_BLKASTQED bit was not set.
20 A5 D0 088A 2407 MOVL LKB$S_BLKASTADR(R5),- ; Store blocking AST address
10 A5 088D 2408 LKB$S_AST(R5)
18 A5 FE77 CF 9E 088F 2409 MOVAB LOCK_KAST,LKB$S_KAST(R5); Store address of kernel AST routine
```

```
52 02 9A 0895 2410 MOVZBL #PRI$ RESAVL,R2 ; Store priority increment class
00000000'GF 16 0898 2411 JSB G^SCH$QAST ; Queue the AST
05 089E 2412 70$: RSB
089F 2413
089F 2414 :++
089F 2415 : Subroutine to call blocking subroutine for system owned locks
089F 2416 :
089F 2417 : Input:
089F 2418 : R5 Address of LKB
089F 2419 : Output:
089F 2420 : R0 - R5 not preserved
089F 2421 :--
089F 2422
089F 2423 CALL_BLK SUBR:
2A 02 AA 089F 2424 BICW #LKBSM_DBLKAST,- ; Clear deliver blocking AST bit
A5 08A1 2425 LKBSW STATUS(R5)
08 AB 08A3 2426 BISW #LKBSM_BLKASTQED,- ; Set blocking AST queued bit
2A A5 08A5 2427 LKBSW STATUS(R5)
51 14 A5 D0 08A7 2428 MOVL LKBSL_ASTPRM(R5),R1 ; Get AST parameter
08AB 2429
08AB 2430 ; Call blocking subroutine. R5 points to LKB, R1 contains AST
08AB 2431 ; parameter. IPL is at IPL$_SYNCH. R0 - R5 need not be
08AB 2432 ; preserved by subroutine.
08AB 2433
20 B5 16 08AB 2434 JSB @LKBSL_BLKASTADR(R5) ; Call blocking subroutine
05 08AE 2435 RSB
08AF 2436
08AF 2437 .DSABL LSB
```

```
08AF 2439 .SBTTL LCK$COMP_GGMODE - Compute group grant mode
08AF 2440
08AF 2441 :++
08AF 2442 : FUNCTIONAL DESCRIPTION:
08AF 2443 :
08AF 2444 : This routine computes the group grant mode for a particular
08AF 2445 : resource. It does this by maximizing the granted lock modes
08AF 2446 : for all locks in the granted and conversion queues. Note that
08AF 2447 : this routine is often called with the lock of interest (to possibly
08AF 2448 : be granted) not on any queue.
08AF 2449 :
08AF 2450 : CALLING SEQUENCE:
08AF 2451 :
08AF 2452 : BSBW LCK$COMP_GGMODE
08AF 2453 : (Note IPL must be at IPL$_SYNCH)
08AF 2454 :
08AF 2455 : INPUTS:
08AF 2456 :
08AF 2457 : R8 Address of RSB
08AF 2458 :
08AF 2459 : OUTPUTS:
08AF 2460 :
08AF 2461 : R5 Group grant mode
08AF 2462 :
08AF 2463 : COMPLETION CODES:
08AF 2464 :
08AF 2465 : None
08AF 2466 :
08AF 2467 : SIDE EFFECTS:
08AF 2468 :
08AF 2469 : R0 and R2 are destroyed
08AF 2470 : Note: R1 must be preserved
08AF 2471 :--
08AF 2472 :
08AF 2473 : .IF NDF LOADSW
000008AF 2474 : .PSECT LOCKMGR
08AF 2475 : .ENDC
08AF 2476 :
08AF 2477 LCK$COMP_GGMODE::
08AF 2478
08AF 2479 ASSUME RSB$$_CVTQFL EQ RSB$$_GRQFL+8
08AF 2480
08AF 2481 CLRL R5 ; Initialize group grant mode
50 10 A8 DE 08B1 2482 MOVAL RSB$$_GRQFL(R8),R0 ; Get address of granted queue
08AF 2483 BSBW 10$ ; Compute g.g. mode for that queue
50 08 C0 08B7 2484 ADDL #8,R0 ; Get address of conversion queue
08AF 2485 08BA ; fall through to ...
08AF 2486 08BA
08AF 2487 ; Subroutine to compute group grant mode for a single queue
08AF 2488 08BA
52 50 D0 08BA 2489 10$: MOVL R0,R2 ; Address of queue header
52 62 D0 08BD 2490 20$: MOVL (R2),R2 ; Get address of next element
50 52 D1 08C0 2491 CMPL R2,R0 ; Reached queue header yet?
08AF 2492 08C3 BEQL 30$ ; Yes
55 FD A2 91 08C5 2493 CMPB LKB$$_GRMODE-LKB$$_SQFL(R2),R5 ; Granted mode greater
08AF 2494 08C9 ; than group grant mode?
08AF 2495 08C9 BLEQU 20$ ; No, continue down queue
```


SYSENQDEQ
V04-000

- ENQUEUE/DEQUEUE SYSTEM SERVICES^{K 2}
LCK\$COMP_GGMode - Compute group grant mo

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 54
(18)

SY
VO

55 FD A2 9A 08CB 2496 MOVZBL LKBSB_GRMODE-LKBSL_SQFL(R2),R5 ; Yes, granted mode becomes
08CF 2497 ; group grant mode
EC 11 08CF 2498 BRB 20\$; continue down queue
05 08D1 2499 30\$: RSB

```
08D2 2501      .SBTTL LCK$GRANTCVTS - Grant conversions
08D2 2502      .SBTTL LCK$GRANTWTRS - Grant waiters
08D2 2503
08D2 2504      ++
08D2 2505      : FUNCTIONAL DESCRIPTION:
08D2 2506      :
08D2 2507      : These two routines try to grant waiting conversions or
08D2 2508      : waiting new locks. They are called when another lock
08D2 2509      : is dequeued or converted if there is a possibility that a waiting
08D2 2510      : lock request may be granted.
08D2 2511
08D2 2512      : CALLING SEQUENCE:
08D2 2513      :
08D2 2514      : BSBW LCK$GRANTCVTS (or LCK$GRANTWTRS)
08D2 2515      : (Note: IPL must be at IPL$_SYNCH)
08D2 2516
08D2 2517      : INPUT PARAMETERS:
08D2 2518      :
08D2 2519      : R5 Contains group grant mode
08D2 2520      : R8 Address of RSB
08D2 2521
08D2 2522      : OUTPUT PARAMETERS:
08D2 2523      :
08D2 2524      : None
08D2 2525
08D2 2526      : SIDE EFFECTS:
08D2 2527      :
08D2 2528      : R0 - R4 are destroyed
08D2 2529      : --
08D2 2530
08D2 2531      .IF NDF LOADSW
0000 08D2 2532      .PSECT LOCKMGR
08D2 2533      .ENDC
08D2 2534
08D2 2535      .ENABL LSB
08D2 2536
08D2 2537 LCK$GRANTCVTS::
08D2 2538      PUSH R6 ; Save R6
56 18 B8 0F 08D4 2539 10$: REMQUE @RSB$_CVTQFL(R8),R6 ; Remove head of conversion queue
56 43 1D 08D8 2540      BVS 70$ ; Nothing on conversion queue
56 38 C2 08DA 2541      SUBL #LKB$_SQFL,R6 ; Position R6 to start of LKB
00000000 GF 95 08DD 2542      TSTB G^LKB$_STALLREQS ; Are we stalling requests?
51 34 A6 9A 08E3 2543      BNEQ 65$ ; Yes
10 F711 CF45 51 E0 08E5 2544 20$: MOVZBL LKB$_RMODE(R6),R1 ; Get requested mode of conversion
55 35 A6 91 08E9 2545      BBS R1,LCK$COMPAT TBL[R5],40$ ; Branch if compat. to grant conversion
FFB6 30 08F4 2546      CMPB LKB$_GRMODE(R6),R5 ; Is granted mode = g.g. mode?
05 F701 CF45 51 E1 08F6 2547      BNEQ 60$ ; No
FD08 30 08F9 2548      BSBW LCK$COMP GGMODE ; Yes, try recomputing g.g. mode
OD A8 55 90 08F9 2549      BBC R1,LCK$COMPAT TBL[R5],55$ ; Branch if not compatible
38 A6 OE 0900 2550 40$: BSBW LCK$GRANT_LOCR ; Grant this conversion
18 A8 56 8ED0 2551      BRB 10$ ; Try the next conversion too
8ED0 05 0903 2552 55$: MOV R5,RSB$_CGMODE(R8) ; Store conversion grant mode
0905 2553 60$: INSQUE LKB$_SQFL(R6),- ; Insert lock at head of
090C 2554      RSB$_CVTQFL(R8) ; conversion queue
090E 2555      POPL R6 ; Restore R6
0911 2556      RSB
0912 2557
```

```
0912 2558 65$: ; We are stalling some requests - see which ones
0912 2559
F5 19 0912 2560 BLSS 60$ ; We are stalling all requests
09 09 0914 2561 BBC #LKB$V_PROTECT,- ; We are only stalling protected locks
CC 2A A6 0916 2562 LKB$W_STATUS(R6),20$ ; Branch if this is not a protected lock
EE 11 0919 2563 BRB 60$ ; Exit without trying to grant lock
0918 2564
0918 2565 LCK$GRANTWTRS::
0918 2566
0918 2567 ; Get here if the conversion queue is empty.
0918 2568 ; Try granting waiting locks. Group grant mode is in R5.
0918 2569 ; This routine is different than the one above because we
0918 2570 ; must skip over locks in any SCS wait state.
0918 2571
56 DD 0918 2572 PUSHL R6 ; Save R6
57 DD 091D 2573 PUSHL R7 ; Save R7
56 20 A8 9E 091F 2574 MOVAB RSB$W_QTQFL(R8),R6 ; Get address of wait queue
57 56 D0 0923 2575 MOVL R6,R7 ; Save in R7
0926 2576
56 66 D0 0926 2577 75$: MOVL (R6),R6 ; Get next LKB
57 56 D1 0929 2578 80$: CMPL R6,R7 ; Reached the end?
2A 13 092C 2579 BEQL 90$ ; Yes
FF 8F 91 092E 2580 CMPB #LKB$K_WAITING,- ; Is the state = WAITING
FE A6 0931 2581 LKB$B_STATE-LKB$W_QTQFL(R6)
F1 12 0933 2582 BNEQ 75$ ; No, skip over it
00000000 GF 95 0935 2583 TSTB G^LCK$GB_STALLREQS ; Are we stalling requests?
22 12 0938 2584 BNEQ 95$ ; Yes
51 FC A6 9A 093D 2585 85$: MOVZBL LKB$B_RQMODE-LKB$W_QTQFL(R6),R1 ; Get requested mode
10 F6B9 CF45 71 E1 0941 2586 BBC R1,LCK$COMPAT_TBL[R5],90$ ; Branch if incompatible
66 DD 0948 2587 PUSHL (R6) ; Save address of next LKB
50 66 OF 094A 2588 REMQUE (R6),R0 ; Remove from wait queue
56 38 C2 094D 2589 SUBL #LKB$W_QTQFL,R6 ; Position R6 to start of LKB
FCB8 30 0950 2590 BSBW LCK$GRANT_LOCK ; Grant this lock
56 8ED0 0953 2591 POPL R6 ; Resume queue where we left off
D1 11 0956 2592 BRB 80$ ; Try next waiting lock
0958 2593
57 8ED0 0958 2594 90$: POPL R7 ; Restore R7
56 8ED0 095B 2595 POPL R6 ; Restore R6
05 095E 2596
095F 2597
095F 2598 95$: ; We are stalling some requests - see which ones
095F 2599
F7 19 095F 2600 BLSS 90$ ; We are stalling all requests
09 09 0961 2601 BBC #LKB$V_PROTECT,- ; Branch if this is not a protected lock
D7 F2 A6 0963 2602 LKB$W_STATUS-LKB$W_QTQFL(R6),85$
F0 11 0966 2603 BRB 90$ ; Exit without trying to grant lock
0968 2604
0968 2605 .DSABL LSB
```

```
0968 2607 .SBTTL VERIFYLOCKID - Verify lock id
0968 2608
0968 2609 :++
0968 2610 : FUNCTIONAL DESCRIPTION:
0968 2611 :
0968 2612 : VERIFYLOCKID verifies a lock id for correct process ownership
0968 2613 : and access mode and then converts it into a LKB address.
0968 2614 :
0968 2615 : VERIFYPARLOCKID is similar except that the lockid is that
0968 2616 : of a parent lock and the access mode checking is different.
0968 2617 :
0968 2618 : CALLING SEQUENCE:
0968 2619 :
0968 2620 : BSBW VERIFYLOCKID
0968 2621 : Note: IPL must be at IPL$ SYNCH to verify that an LKB is
0968 2622 : not deallocated while it is being verified. After this
0968 2623 : routine returns, it is permissible to lower IPL to
0968 2624 : IPL$ASTDEL because the process that owns this lock
0968 2625 : cannot interrupt to dequeue it (as long as we're
0968 2626 : not dealing with a system owned lock).
0968 2627 :
0968 2628 : INPUT PARAMETERS:
0968 2629 :
0968 2630 : R1 Lock id
0968 2631 : R4 Address of PCB
0968 2632 :
0968 2633 : OUTPUT PARAMETERS:
0968 2634 :
0968 2635 : R0 Completion code
0968 2636 : R1 Access mode of caller (on success only)
0968 2637 : R6 Address of LKB
0968 2638 :
0968 2639 : COMPLETION CODES:
0968 2640 :
0968 2641 : SS$_NORMAL Lock id was valid and converted to LKB address
0968 2642 : SS$_IVLOCKID Invalid lock id
0968 2643 :
0968 2644 : SIDE EFFECTS:
0968 2645 :
0968 2646 : R0 is destroyed
0968 2647 : --
0968 2648 :
0968 2649 : .IF NDF LOADSW
00000968 2650 : .PSECT LOCKMGR
0968 2651 : .ENDC
0968 2652 :
0968 2653 : .ENABL LSB
0968 2654 :
0968 2655 : ASSUME LKBSV_MODE EQ 0
0968 2656 : ASSUME LKBSS_MODE EQ 2
0968 2657 :
0968 2658 VERIFYLOCKID:
01 DD 0968 2659 PUSHL #1
02 11 096A 2660 BRB 5$
096C 2661
096C 2662 VERIFYPARLOCKID:
00 DD 096C 2663 PUSHL #0
```

```

      56 51 3C 096E 2664 5$: MOVZWL R1,R6 ; Put lockid index in R6
00000000'GF 56 D1 0971 2665 CMPL R6,G^LCK$GL_MAXID ; Is the lock id too big?
      48 1A 0978 2666 BGTRU 30$ ; Yes
50 00000000'GF 10 097A 2667 MOVL G^LCK$GL_IDTBL,R0 ; *** May combine with next instr.
      56 60 46 D0 0981 2668 MOVL (R0)[R6],R6 ; Get LKB address
      3B 18 0985 2669 BGEQ 30$ ; Unallocated id
      30 A6 51 D1 0987 2670 CMPL R1,LKB$L_LKID(R6) ; Check sequence number
      35 12 098B 2671 BNEQ 30$ ; Not valid
      50 DC 098D 2672 MOVPSL R0 ; Get current PSL
      16 EF 098F 2673 EXTZV #PSL$V_PRVMOD,- ; Extract previous mode field
50 51 50 02 0991 2674 #PSL$S_PRVMOD,R0,R1
      OB A6 FC 8F 8B 0994 2675 BICB3 #^XFC,[KBSB_RMOD(R6),R0 ; Get access mode of lock
      6E D5 099A 2676 TSTL (SP) ; Determine which acmode checks to make
      07 13 099C 2677 BEQL 10$ ; Parent checks
      50 51 91 099E 2678 CMPB R1,R0 ; Caller have privilege to access lock?
      11 1B 09A1 2679 BLEQU 15$ ; Yes
      1D 11 09A3 2680 BRB 30$ ; No
      50 51 91 09A5 2681 10$: CMPB R1,R0 ; Only less priv. modes can be sub-locks
      18 1F 09A8 2682 BLSSU 30$ ; to more priv. modes.
      50 50 A6 D0 09AA 2683 MOVL LKB$L_RSB(R6),R0 ; Get RSB address
      4E A0 51 91 09AE 2684 CMPB R1,RSB$B_RMOD(R0) ; Caller have privilege to access res.?
      0E 1A 09B2 2685 BGTRU 30$ ; No
      0C A6 D1 09B4 2686 15$: CMPL LKB$L_PID(R6),- ; Compare LKB PID with current
      60 A4 09B7 2687 PCB$L_PID(R4) ; processes' PID
      10 12 09B9 2688 BNEQ 50$ ; Somebody else's id (or 0)
      09BB 2689
      5E 04 C0 09BB 2690 20$: ADDL #4,SP ; Remove flag
      50 01 3C 09BE 2691 MOVZWL S^#SS$_NORMAL,R0 ; Success
      05 09C1 2692 RSB
      09C2 2693
      50 5E 04 C0 09C2 2694 30$: ADDL #4,SP ; Remove flag
      2124 8F 3C 09C5 2695 MOVZWL #SS$_IVLOCKID,R0 ; Invalid lock id
      05 09CA 2696 RSB
      09CB 2697
      09CB 2698 50$: ; The caller's PID and the lock's PID don't match. If this is not
      09CB 2699 ; a master copy lock, then it may be a system-owned lock. If it
      09CB 2700 ; is system-owned and the caller's access mode is EXEC or KERNEL
      09CB 2701 ; the he can reference this lock. Otherwise, it is an error.
      09CB 2702 ; We determine if it is system-owned via two different methods
      09CB 2703 ; depending on whether we are verifying a parent lock or not.
      09CB 2704 ; If it is a parent lock, we just need to determine if the CVTSYS
      09CB 2705 ; flag is set. Otherwise, we need to verify that the PID is zero.
      09CB 2706 ; This is because even system owned locks go through transient
      09CB 2707 ; states when the PID is not zero (e.g. conversion in progress
      09CB 2708 ; and lock is mastered on another node). If a system owned lock
      09CB 2709 ; is in this state (PID non-zero) then it cannot be manipulated
      09CB 2710 ; by another process. But it can be used as a parent for other
      09CB 2711 ; system owned locks.
      09CB 2712
      2A 10 B3 09CB 2713 BITW #LKB$M_MSTCPY,- ; Is this another system's master copy?
      A6 09CD 2714 LKB$W_STATUS(R6)
      F1 12 09CF 2715 BNEQ 30$ ; Yes, error
      6E D5 09D1 2716 TSTL (SP) ; No, choose system ownership check
      07 13 09D3 2717 BEQL 60$ ; Parent lock check
      0C A6 D5 09D5 2718 TSTL LKB$L_PID(R6) ; Is this a system lock?
      07 03 09D8 2719 BEQL 70$ ; Yes
      E6 11 09DA 2720 BRB 30$ ; No, error
```

E1 28	06	E1	09DC	2721	60\$:	BBC	#LCK\$V CVTSYS,		; Branch if not system owned
01	A6		09DE	2722			LKBSW FLAGS(R6),	30\$	
	S1	91	09E1	2723	70\$:	CMPB	R1, #PSL\$C_EXEC		; Yes, is the caller privileged?
	D5	1B	09E4	2724		BLEQU	20\$		; Yes
	DA	11	09E6	2725		BRB	30\$		; No
			09E8	2726					
			09E8	2727		.DSABL	LSB		

```
09E8 2729      .SBTTL  EXE$DEQ - Dequeue system service
09E8 2730
09E8 2731      :++
09E8 2732      : FUNCTIONAL DESCRIPTION:
09E8 2733      :
09E8 2734      :     This routine handles the $DEQ system service
09E8 2735
09E8 2736      : CALLING SEQUENCE:
09E8 2737      :
09E8 2738      :     CALLS/G EXE$DEQ (Actually called through the system service
09E8 2739      :     dispatcher)
09E8 2740
09E8 2741      : INPUT PARAMETERS:
09E8 2742      :
09E8 2743      :     LOCKID(AP)      Lock id
09E8 2744      :     VALBLK(AP)     Address of value block
09E8 2745      :     DEQ_ACMODE(AP) Access mode of locks to dequeue (only used if
09E8 2746      :     DEQALL flag is set)
09E8 2747      :     DEQ_FLAGS(AP)   Flags
09E8 2748
09E8 2749      :     R4              Address of PCB
09E8 2750
09E8 2751      : OUTPUT PARAMETERS:
09E8 2752      :
09E8 2753      :     R0              Completion code
09E8 2754
09E8 2755      : COMPLETION CODES:
09E8 2756      :
09E8 2757      :     $$_NORMAL      Successful completion
09E8 2758      :     $$_IVLOCKID    Invalid lock id
09E8 2759      :     $$$_ACCVIO     Access violation (on VALBLK)
09E8 2760      :     $$$_SUBLOCKS   Lock has sublocks
09E8 2761      :     $$$_CANCELGRANT Cannot cancel a granted lock
09E8 2762      :--
09E8 2763
0000 0111 2764      .IF NDF LOADSW
0111 2765      .PSECT  Y$EXEPAGED
0111 2766      .ENDC
0111 2767
0111 2768      .ENABL  LSB
0111 2769
0111 2770
0111 2771      .IF NDF LOADSW
OFFC 0111 2772      .ENTRY  EXE$DEQ,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
0113 2773      .IFF
0113 2774      .ENTRY  EXE$$DEQ,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
0113 2775      .ENDC
0113 2776
0113 2777      ; First see if this is a dequeue of a specific lock or a dequeue
0113 2778      ; of all locks at the specified access mode (maximized with caller's
0113 2779      ; access mode) and outer modes.
0113 2780
0113 2781      ASSUME  LCK$V_DEQALL  EQ  0
0113 2782
51   04 AC  D0 0113 2783      MCL  LOCKID(AP),R1      ; Get lock id
55   10 AC  3C 0117 2784      MOVZWL DEQ_FLAGS(AP),R5      ; Get dequeue flags
56   55  E8 011B 2785      BLBS  R5,DEQ_ALL      ; Branch if dequeue all
```

```
011E 2786
011E 2787 ; It's a dequeue of a specific lock
011E 2788
59 08 AC D0 011E 2789 MOVL VALBLK(AP),R9 ; Get address of value block
10 13 0122 2790 BEQL 20$ ; No value block
0124 2791 IFNORD #16,(R9),25$ ; Branch if value block not readable
7E 08 A9 7D 012A 2792 MOVQ 8(R9),-(SP) ; Push value block onto stack
7E 69 7D 012E 2793 MOVQ (R9),-(SP)
59 5E D0 0131 2794 MOVL SP,R9 ; R9 points to value block
0134 2795 20$: SETIPL 95$ ; Raise IPL to IPL$ SYNCH and
0138 2796 ; lock pages in memory
082A' 30 013B 2797 BSBW VERIFYLOCKID ; Verify lock id and return LKB in R6
1D 5C E9 013E 2798 BLBC R0,DEQ_EXIT ; Error
0141 2799
0141 2800 ; Check to see if we are stalling all requests
0141 2801
00000000'GF 95 0141 2802 TSTB G^LCK$GB_STALLREQS ; Are we stalling all requests?
19 19 0147 2803 BLSS 22$ ; Yes
0149 2804
0149 2805 ; LKB address is in R6. Value block address (or 0) is in R9.
0149 2806 ; Dequeue the lock and grant any waiting locks.
0149 2807
00000002 00000002 0149 2808 .IF NE CAS MEASURE
00000000'GF D6 0149 2809 INCL G^PMSS$GL_DEQ_LOC
014F 2810 .ENDC
014F 2811
54 55 D0 014F 2812 MOVL R5,R4 ; Move flags
57 D4 0152 2813 CLRL R7 ; Use default status
092E' 30 0154 2814 BSBW LCK$DEQLOCK ; Dequeue the lock
0124 8F 50 B1 0157 2815 CMPW R0,#SS$_INSFMEM ; Handle insufficient memory
07 13 015C 2816 BEQL 23$
015E 2817
015E 2818 DEQ_EXIT:
015E 2819 ; Exit $DEQ system service. Status should already be in R0.
015E 2820
015E 2821 SETIPL #IPL$_ASTDEL ; Lower IPL
04 0161 2822 RET
0162 2823
082B' 31 0162 2824 22$: BRW STALL_REQ ; Stall request
081A' 31 0165 2825 23$: BRW WAIT_FOR_POOL ; Wait for pool
0168 2826
50 0C 3C 0168 2827 25$: MOVZWL S^#SS$_ACCVIO,R0 ; Access violation
F1 11 016B 2828 BRB DEQ_EXIT
016D 2829
50 212C 8F 3C 016D 2830 35$: MOVZWL #SS$_SUBLOCKS,R0 ; System lock with sublocks
EA 11 0172 2831 BRB DEQ_EXIT
0174 2832
0174 2833 DEQ_ALL:
0174 2834 ; Dequeue all locks at the specified access mode (maximized
0174 2835 ; with caller's mode) and less privileged modes. Since this list
0174 2836 ; is normally kept in the order locks were taken out, one pass
0174 2837 ; through the list will normally be able to dequeue all the
0174 2838 ; specified locks. However, two things may cause the list to
0174 2839 ; be out of order. First of all, waiting locks are kept at the
0174 2840 ; end of list (for the convenience of deadlock detection) and
0174 2841 ; secondly, if a lock with sublocks is converted (and must wait)
0174 2842 ; it ends up out of order on the list. If the list is out of order,
```



```
0174 2843 ; the result is that we get $$$ SUBLOCKS errors when we try to
0174 2844 ; dequeue out of order locks. It is therefore necessary to count
0174 2845 ; these errors and if there are any to repeat the loop, again
0174 2846 ; trying to dequeue all the specified loops. In order to guarantee
0174 2847 ; eventual completion we make sure that each time the loop is repeated,
0174 2848 ; the number of $$$ SUBLOCKS errors is less than the previous time.
0174 2849
0174 2850 ASSUME LKBSV_MODE EQ 0
0174 2851 ASSUME LKBSV_MODE EQ 2
0174 2852
55 02 AA 0174 2853 BICW #LCKSM_CANCEL,R5 ; Don't allow CANCEL if DEQALL is set
55 55 DD 0177 2854 PUSHL R5 ; Save dequeue flags
56 51 D0 0179 2855 MOVL R1,R6 ; Move parent lock id
15 13 017C 2856 BEQL 40$ ; No parent - dequeue all locks
00000968'EF 16 0185 2857 SETIPL 95$ ; Raise IPL to verify lockid
D0 50 E9 018B 2858 JSB VERIFYLOCKID ; Convert to LKB address
OC A6 D5 018E 2859 BLBC R0,DEQ_EXIT ; Error
DA 13 0191 2860 TSTL LKBSL_PID(R6) ; Verify the lock is not system owned
56 DD 0193 2861 BEQL 35$ ; It is - error
40$: 0195 2862 PUSHL R6 ; Save LKB address or zero
0195 2863 SETIPL #IPL$ ASTDEL ; Lower IPL to allow page faults
50 OC AC FFFFFFFC 8F CB 0198 2864 BICL3 #^C<35,DEQ_ACMODE(AP),R0 ; Get specified access mode
00000000'GF 16 01A1 2865 JSB G^EXESMAXACMODE ; Maximize with previous mode
7E 01 CE 01A7 2866 MNEGL #1,-(SP) ; Initialize last error count to -1
00 DD 01AA 2867 PUSHL #0 ; Initialize current error count to 0
50 DD 01AC 2868 PUSHL R0 ; Save access mode
59 D4 01AE 2869 CLRL R9 ; Indicate no value block
SA 0104 C4 DE 01B0 2870 MOVAL PCB$ LOCKQFL(R4),R10 ; Get address of PCB lock queue head
5B 5A D0 01B5 2871 MOVL R10,RT1 ; Save in R11
01B8 2872
01B8 2873 50$: ; Process next LKB in list. It's friendly to lower IPL to IPL$ ASTDEL
01B8 2874 ; so that system events can occur. Otherwise we could dequeue
01B8 2875 ; thousands of locks staying at IPL$ SYNCH the entire time.
01B8 2876 ; But lowering to IPL$ ASTDEL exposes us to at least two race
01B8 2877 ; conditions. The first is that we could have the address of
01B8 2878 ; an LKB in R6 (at IPL$ ASTDEL) and deadlock detection could
01B8 2879 ; dequeue the lock out from underneath us. The solution to this
01B8 2880 ; is to stay at IPL$ SYNCH between fetching the LKB address and
01B8 2881 ; dequeuing it. The other race condition is that by lowering
01B8 2882 ; to IPL$ ASTDEL, a waiting lock could be granted, this moving
01B8 2883 ; from the tail of the list to the head. Since we traverse this
01B8 2884 ; list from the head to the tail, we could miss this lock. The
01B8 2885 ; solution to this one is to verify that the head of the list
01B8 2886 ; doesn't change while we lower and raise IPL.
01B8 2887
56 6B D0 01B8 2888 MOVL (R11),R6 ; Get first LKB on list
01B8 2889 SETIPL #IPL$ ASTDEL ; Lower IPL to allow system events
01BE 2890 SETIPL 95$ ; Raise IPL and lock pages in memory
6B 56 D1 01C5 2891 CMPL R6,(R11) ; Did the head of the queue change?
56 6A 12 01C8 2892 BNEQ 75$ ; Yes, start over again
5B 56 D0 01CA 2893 MOVL (R10),R6 ; Get next LKB in list
51 51 D1 01CD 2894 CMPL R6,R11 ; Reached end of list?
56 C0 A6 DE 01D2 2895 BEQL 70$ ; Yes
50 OB A6 FC 8F 8B 01D6 2896 MOVAL -LKBSL_OWNQFL(R6),R6 ; Back up R6 to point to start of LKB
6E 50 91 01DC 2897 BICB3 #^XFC,[KBSB_RMOD(R6),R0 ; Get lock access mode
3D 1F 01DF 2898 CMPB R0,(SP) ; Is lock access mode < spec. mode?
2899 BLSSU 60$ ; Yes, don't dequeue
```

```
51 0C AE D0 01E1 2900      MOVL 12(SP),R1      ; Get parent LKB address
    13 13 01E5 2901      BEQL 55$      ; No parent - dequeue all locks
    01E7 2902
    01E7 2903      ; Have a candidate lock to dequeue (in R6). Exit the loop
    01E7 2904      ; if the specified parent lock (in R1) has a zero reference count.
    01E7 2905      ; Otherwise, see if our candidate lock (in R6) is a sublock of our
    01E7 2906      ; parent lock (in R1).
    01E7 2907
    4C A1 B5 01E7 2908      TSTW LKBSW_REFCNT(R1)      ; Are all its sublocks gone?
    51 13 01EA 2909      BEQL 80$      ; Yes
    50 56 D0 01EC 2910      MOVL R6,R0      ; R0 will point to each LKB up the tree
    50 48 A0 D0 01EF 2911 53$:      MOVL LKBSL_PARENT(R0),R0      ; Move up one level in tree
    29 13 01F3 2912      BEQL 60$      ; Reached the top without a match
    51 50 D1 01F5 2913      CMPL R0,R1      ; Does this match specified parent?
    F5 12 01F8 2914      BNEQ 53$      ; No, keep going up tree
    01FA 2915
    01FA 2916 55$:      ; Have an LKB (in R6) to be dequeued.
    01FA 2917
    00000002 01FA 2918      .IF NE CAS MEASURE
00000000'GF D6 01FA 2919      INCL G^PMSSGL_DEQ_LOC
    0200 2920      .ENDC
    0200 2921
    54 10 AE D0 0200 2922      MOVL 16(SP),R4      ; Fetch dequeue flags
    57 D4 0204 2923      CLRL R7      ; Use default status
00000000'GF 95 0206 2924      TSTB G^LCK$GB_STALLREQS      ; Are we stalling all requests?
    38 19 020C 2925      CLSS 85$      ; Yes
    0874' 30 020E 2926      BSBW LCK$DEQLOCK      ; Dequeue it
    A4 50 E8 0211 2927      BLBS R0,50$      ; Branch on success
    0214 2928
0124 8F 50 B1 0214 2929      CMPW R0,#SS$_INSFMEM      ; Check for insufficient memory
    28 13 0219 2930      BEQL 83$
    021B 2931
    04 AE D6 021B 2932      INCL 4(SP)      ; Increment error count
    5A 6A D0 021E 2933 60$:      MOVL (R10),R10      ; Skip this LKB
    95 11 0221 2934      BRB 50$
    0223 2935
    0223 2936 70$:      ; Completed a loop through all of the process's locks. If the
    0223 2937      ; current error count is zero, then we're done. If it's non-zero
    0223 2938      ; then we have to run through the list again as long as the error
    0223 2939      ; count this time was less than the error count last time.
    0223 2940
    04 AE D5 0223 2941      TSTL 4(SP)      ; Test current error count
    15 13 0226 2942      BEQL 80$      ; Zero - all done.
08 AE 04 AE D1 0228 2943      CMPL 4(SP),8(SP)      ; Compare current count with last one
    1A 1E 022D 2944      BGEQU 90$      ; Bugcheck if it didn't go down
08 AE 04 AE D0 022F 2945      MOVL 4(SP),8(SP)      ; Current count becomes previous count
    04 AE D4 0234 2946 75$:      CLRL 4(SP)      ; Zero current count
    5A 5B D0 0237 2947      MOVL R11,R10      ; R10 now points to list head again
    FF7B 31 023A 2948      BRW 50$      ; Repeat the loop
    023D 2949
    50 01 3C 023D 2950 80$:      MOVZWL S^#SS$_NORMAL,R0      ; Set completion status
    FF1B 31 0240 2951      BRW DEQ_EXIT
    0243 2952
    0A3C' 31 0243 2953 83$:      BRW WAIT_FOR_POOL      ; Wait for pool
    0A47' 31 0246 2954 85$:      BRW STALL_REQ      ; Stall request
    0249 2955
    0249 2956 90$:      BUG_CHECK DEQSUBLOCKS,FATAL
```

SYSENQDEQ
V04-000

- ENQUEUE/DEQUEUE SYSTEM SERVICES^{H 3}
EXESDEQ - Dequeue system service

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 64
(21)

SYS
V04

```
00000008 0240 2957  
          0240 2958 95$: .LONG IPL$ SYNCH          ; End of locked down code  
          0251 2959      ASSUME .-20$ LE 512        ; Must be on adjacent pages  
          0251 2960      ASSUME .-DEQ_ALL LE 512  
          0251 2961  
          0251 2962      .DSABL LSB
```

```
0251 2964 .SBTTL LCK$CANCEL_CVT - Cancel a waiting conversion
0251 2965
0251 2966 :++
0251 2967 : FUNCTIONAL DESCRIPTION:
0251 2968 :
0251 2969 : This routine cancels waiting conversions instead of dequeuing them.
0251 2970 : This means the lock is inserted back on the granted queue at it's
0251 2971 : old mode. However, if it was at the head of the conversion queue,
0251 2972 : then we have to try granting other conversions. Note that canceled
0251 2973 : locks get their old blocking AST address restored. Also, if the lock
0251 2974 : was formerly a system owned lock, then the CVTTOSYS bit is set so that
0251 2975 : after the completion AST is delivered, the lock is converted back
0251 2976 : to a system owned lock.
0251 2977
0251 2978 : CALLING SEQUENCE:
0251 2979 :
0251 2980 : BSBW LCK$CANCEL_CVT
0251 2981 : IPL must be at IPL$_SYNCH
0251 2982
0251 2983 : INPUTS:
0251 2984 :
0251 2985 : R6 Address of LKB
0251 2986 : R7 Final completion status to store in LKB$$_LKST1 or 0
0251 2987 : in which case $$$_CANCEL is used (not needed if this is a
0251 2988 : master copy lock)
0251 2989 : R8 Address of RSB
0251 2990
0251 2991 : OUTPUTS:
0251 2992 :
0251 2993 : None
0251 2994
0251 2995 : SIDE EFFECTS:
0251 2996 :
0251 2997 : R0 - R4 and R9 are destroyed
0251 2998 :--
0251 2999
0251 3000 .IF NDF LOADSW
000009E8 3001 .PSECT LOCKMGR
09E8 3002 .ENDC
09E8 3003
09E8 3004 LCK$CANCEL_CVT::
09E8 3005 SUBL3 #LKB$$_SQFL,- : Save address of lock at the
59 18 A8 C3 09EA 3006 RSB$$_CVTQFL(R8),R9 : head of conversion queue
51 38 A6 0F 09ED 3007 REMQUE LKB$$_SQFL(R6),R1 : Remove this lock
17 2A A6 E0 09F1 3008 BBS #LKB$$_MSTCPY,- : Skip "process" code if this is a
2C A6 57 D0 09F3 3009 LKB$$_STATUS(R6),10$ : master copy
0830 8F 3C 09FA 3010 MOVL R7,LKB$$_LKST1(R6) : Store specified status
2C A6 06 12 09FC 3011 BNEQ $$ : Had one
06 2A A6 E1 0A00 3012 MOVZWL #$$$_CANCEL,- : Use default status instead
0100 8F A8 0A02 3013 LKB$$_LKST1(R6)
2A A6 57 0A04 3014 5$: BBC #LKB$$_WASSYSOWN,- : Branch if lock wasn't system owned
51 35 A6 9A 0A07 3015 LKB$$_STATUS(R6),10$
2A A6 57 0A08 3016 BBS #LKB$$_CVTTOSYS,- : Lock should be cvted to system owned
51 35 A6 9A 0A0D 3017 LKB$$_STATUS(R6) : after completion AST is delivered
2A A6 57 0A0B 3018 10$: MOVL LKB$$_OLDBLKAST(R6),- : Restore old blocking AST address
51 35 A6 9A 0A10 3019 LKB$$_BLKASTADR(R6)
2A A6 57 0A12 3020 MOVZBL LKB$$_GRMODE(R6),R1 : Get current granted mode
```

```
38 A8 D5 0A16 3021 TSTL RSB$ _CSID(R8) ; Is this a process copy?
14 12 0A19 3022 BNEQ 30$ ; Yes, skip granting other locks
FC05 30 0A1B 3023 BSBW LCK$REGRANTLOCK ; Regrant this lock
59 56 D1 0A1E 3024 CMPL R6,R9 ; Was it at the head of the queue?
OB 12 0A21 3025 BNEQ 20$ ; No
0A23 3026
0A23 3027 ; We have regranted a lock that was at the head of the conversion
0A23 3028 ; queue. Therefore, it is necessary to try to grant additional
0A23 3029 ; locks. Also, this will reset the conversion grant mode if we
0A23 3030 ; set it incorrectly below.
0A23 3031
55 OC A8 9A 0A23 3032 MOVZBL RSB$B,GGMODE(R8),R5 ; Get group grant mode
OD A8 55 90 0A27 3033 MOVB R5,RSB$B,CGMODE(R8) ; and set conv. grant mode equal to it
FEA4 30 0A2B 3034 BSBW LCK$GRANTCVTS ; Try granting more locks
05 0A2E 3035 20$: RSB
0A2F 3036
FC2B 30 0A2F 3037 30$: BSBW LCK$GRANT_REM ; Regrant this lock
05 0A32 3038 RSB
```

```
0A33 3040      .SBTTL LCK$DEQLOCK - Dequeue a lock
0A33 3041
0A33 3042      :++
0A33 3043      : FUNCTIONAL DESCRIPTION
0A33 3044      :
0A33 3045      : This routine dequeues a specified (by LKB address) lock and
0A33 3046      : grants any waiting locks, if possible. If there are no
0A33 3047      : waiters or holders of the lock, the RSB is deallocated.
0A33 3048
0A33 3049      : CALLING SEQUENCE:
0A33 3050      :
0A33 3051      : BSBW      LCK$DEQLOCK
0A33 3052      : IPL must be at IPL$_SYNCH
0A33 3053
0A33 3054      : INPUT PARAMETERS:
0A33 3055      :
0A33 3056      : R4      Dequeue flags
0A33 3057      : R6      Address of LKB
0A33 3058      : R7      Contains final status to store in LKB$_LKST1 if lock
0A33 3059      : is not granted (i.e. SSS_DEADLOCK) or 0 which indicates
0A33 3060      : a default status should be used (see below).
0A33 3061      : (not needed if this is a master copy lock)
0A33 3062      : R9      Address of value block or 0 if no value block
0A33 3063      : (not needed if this is a CANCEL function)
0A33 3064
0A33 3065      : OUTPUT PARAMETERS:
0A33 3066      :
0A33 3067      : R0      Completion code
0A33 3068
0A33 3069      : COMPLETION CODES:
0A33 3070      :
0A33 3071      : In R0:
0A33 3072      :
0A33 3073      : SSS_NORMAL      Successful completion
0A33 3074      : SSS_SUBLOCKS   Lock has sublocks
0A33 3075      : SSS_CANCELGRANT Cannot cancel a granted lock
0A33 3076      : SSS_INSMEM     Insufficient memory to allocate a CDRP
0A33 3077
0A33 3078      : In LKB$_LKST1 (if R7 = 0)
0A33 3079      :
0A33 3080      : SSS_ABORT - Lock request was aborted
0A33 3081      : SSS_CANCEL - Lock request was canceled
0A33 3082
0A33 3083      : SIDE EFFECTS
0A33 3084      :
0A33 3085      : R0 - R5, and R8 and R9 are clobbered
0A33 3086      : --
0A33 3087
0A33 3088      : .IF NDF LOADSW
00000A33 3089      : .PSECT LOCKMGR
0A33 3090      : .ENDC
0A33 3091
0A33 3092      : .ENABL LSB
0A33 3093
0A33 3094      : INVALID STATE:
0A33 3095      : BUG_CHECK      LOCKMGRERR,FATAL      : Invalid lock state
0A33 3096
```

```
50 212C 8F 3C 0A37 3097 REFCNT_ERROR:
      05 0A37 3098 MOVZWL #SS$_SUBLOCKS,R0
      0A3C 3099 RSB
      0A3D 3100
      0A3D 3101 CANCELGRANT:
      0A3D 3102 MOVZWL #SS$_CANCELGRANT,R0 ; Granted - can't cancel a granted lock
      05 0A42 3103 RSB
      0A43 3104
      36 A6 95 0A43 3105 CANCEL: TSTB LKBS$_STATE(R6) ; Dispatch on lock state
      46 19 0A46 3106 BLSS 6$ ; Waiting - handle as ordinary dequeue
      F3 14 0A48 3107 BGTR CANCELGRANT ; Granted - error
53 38 A8 D0 0A4A 3108 MOVL RBS$_L_CSID(R8),R3 ; Is resource managed by another system?
      0F 13 0A4E 3109 BEQL 3$ ; No
00000000'GF 16 0A50 3110 JSB G^CNX$ALLOC_CDRP ; Alloc. CDRP (and convert CSID to CSB)
      69 50 E9 0A56 3111 BLBC R0,NOCDRP ; None available (or CSID convert error)
00000000'GF 13 0A59 3112 JSB G^LCK$SND_DEQCV ; Send a dequeue message
      87 10 0A5F 3113 3$: BSB LCK$CANCEL_CVT ; Cancel conversion and regrant lock
      50 01 3C 0A61 3114 MOVZWL S^SS$_NORMAL,R0 ; Return success
      05 0A64 3115 RSB
      0A65 3116
      0A65 3117 SNDDEQ_WAIT:
      0A65 3118 ; Need to send a message to master system. Lock is in a
      0A65 3119 ; waiting state.
      0A65 3120
00000000'GF 16 0A65 3121 JSB G^CNX$ALLOC_CDRP ; Alloc. CDRP (and convert CSID to CSB)
      54 50 E9 0A6B 3122 BLBC R0,NOCDRP ; None available (or CSID convert error)
      36 A6 95 0A6E 3123 TSTB LKBS$_STATE(R6) ; Is lock in conversion wait?
      09 13 0A71 3124 BEQL 4$ ; Yes
00000000'GF 16 0A73 3125 JSB G^LCK$SND_DEQWT ; No, send a dequeue message
      00FC 31 0A79 3126 BRW 60$ ; Resume processing
00000000'GF 16 0A7C 3127 4$: JSB G^LCK$SND_DEQCV ; Send a dequeue message
      00F3 31 0A82 3128 BRW 60$
      0A85 3129
      0A85 3130 LCK$DEQLOCK::
      58 50 A6 D0 0A85 3131 MOVL LKBS$_RSB(R6),R8 ; Get RSB address
      54 02 B3 0A89 3132 BITW #LCK$_CANCEL,R4 ; Is CANCEL flag set?
      B5 12 0A8C 3133 BNEQ CANCEL ; Yes
      4C A6 B5 0A8E 3134 6$: TSTW LKBS$_REFCNT(R6) ; Are there any sub locks?
      A4 12 0A91 3135 BNEQ REFCNT_ERROR ; Yes - error
      0A93 3136
      0A93 3137 ASSUME LKBS$_GRANTED EQ 1
      0A93 3138 ASSUME LKBS$_CONVERT EQ 0
      0A93 3139 ASSUME LKBS$_WAITING EQ -1
      0A93 3140
      0A93 3141 ; Dispatch depending on which queue the lock is on and
      0A93 3142 ; whether it is managed remotely
      0A93 3143
50 36 A6 90 0A93 3144 MOVB LKBS$_STATE(R6),R0 ; Get lock state
      60 14 0A97 3145 BGTR DEQ_GRANTED ; Lock is on granted queue
53 38 A8 D0 0A99 3146 MOVL RBS$_L_CSID(R8),R3 ; Is resource managed by another system?
      C6 12 0A9D 3147 BNEQ SNDDEQ_WAIT ; Yes
      50 95 0A9F 3148 TSTB R0 ; What queue is lock on?
      34 13 0AA1 3149 BEQL DEQ_CONVERT ; Lock is on conversion queue
      FF 8F 50 91 0AA3 3150 CMPB R0,#LKBS$_WAITING ; Make sure state = WAITING
      BA 12 0AA7 3151 BNEQ INVALID_STATE ; It's not
      0AA9 3152
      0AA9 3153 DEQ_WAIT:
```

```
0AA9 3154 ; The lock is on the waiting queue. Remove it from the queue and
0AA9 3155 ; see if it was at the head of the queue. If yes, then we may be
0AA9 3156 ; able to grant some locks (but only if the conversion queue is empty).
0AA9 3157 ; If no, then there is no possibility of granting some locks.
0AA9 3158
51 38 A6 0F 0AA9 3159 REMQUE LKBSL_SQFL(R6),R1 ; Remove this lock
10 13 0AAD 3160 BEQL 18$ ; Wait queue is now empty
50 18 A8 DE 0AAF 3161 MOVAL RBSL_CVTQFL(R8),R0 ; See if we can grant any more locks
50 60 D1 0AB3 3162 CMPL (R0),R0 ; Is conversion queue empty?
1C 12 0AB6 3163 BNEQ 5$ ; No - can't grant any other locks
55 0C A8 9A 0AB8 3164 MOVZBL RBSB_GGMode(R8),R5 ; Yes, get group grant mode and
009C 31 0ABC 3165 BRW 40$ ; Try granting some waiters
00B3 31 0ABF 3166 18$: BRW 55$
OAC2 3167
OAC2 3168 NOCDRP: ; Either insufficient memory or CSID conversion error.
OAC2 3169
0124 8F 50 B1 0AC2 3170 CMPW R0,#SS$_INSFMEM ; Is it insufficient memory?
01 12 0AC7 3171 BNEQ 13$ ; No
05 05 0AC9 3172 RSB ; Yes, return error to caller
OACA 3173 13$: BUG_CHECK LOCKMGRERR,FATAL; CSID conversion error
OACE 3174
OACE 3175 SNDDEQ_GRNT:
OACE 3176 ; Need to send a message to master system. Lock is in
OACE 3177 ; granted state.
OACE 3178
00000000'GF 16 0ACE 3179 JSB G^LCK$SND_DEQGR ; Send a dequeue message
00A1 31 0AD4 3180 5$: BRW 60$
OAD7 3181
OAD7 3182 DEQ_CONVERT:
OAD7 3183 ; The lock is on the conversion queue. Remove it from the queue and
OAD7 3184 ; see if it was at the head of the queue. If no, we may be able to
OAD7 3185 ; grant some locks due to the granted mode of this lock going away.
OAD7 3186 ; If yes, we may be able to grant some locks for the same reason
OAD7 3187 ; and for the additional reason of the head of the queue going away.
OAD7 3188
59 38 C3 0AD7 3189 SUBL3 #LKBSL_SQFL,- ; Save address of lock at the
18 A8 0AD9 3190 RBSL_CVTQFL(R8),R9 ; head of conversion queue
51 38 A6 0F 0ADC 3191 REMQUE LKBSL_SQFL(R6),R1 ; Remove this lock
35 A6 91 0AE0 3192 CMPB LKBSB_GMODE(R6),- ; Is lock mode PW or higher?
04 0AE3 3193 #LCK$_PWMode
0C 1F 0AE4 3194 BLSSU 7$ ; No, skip value block processing
54 04 B3 0AE6 3195 BITW #LCK$_INVVALBLK,R4 ; Should value block be invalidated?
07 13 0AE9 3196 BEQL 7$ ; No
02 A8 0AEB 3197 BISW #RBSM_VALINVLD,- ; Yes, invalidate value block
OE A8 0AED 3198 RBSW_STATUS(R8)
3C A8 D6 0AEF 3199 INCL RBSL_VALSEQNUM(R8) ; Increment value block sequence number
59 56 D1 0AF2 3200 7$: CMPL R6,R9 ; Was it the first one on the queue?
70 13 0AF5 3201 BEQL 50$ ; Yes
67 11 0AF7 3202 BRB 45$ ; No
OAF9 3203
OAF9 3204 DEQ_GRANTED:
OAF9 3205 ; The lock is on the granted queue. Remove it from the queue
OAF9 3206 ; and see if it was the only one on the queue. If it was, then see
OAF9 3207 ; if the conversion and wait queues are also empty, and if so then
OAF9 3208 ; the resource block can be deallocated. This situation is special cased
OAF9 3209 ; because it is the normal case. If this lock is not the only one on the
OAF9 3210 ; queue, then see if its granted mode is equal to the group grant mode.
```



```

      OAF9 3211      ; If yes, we may be able to grant some locks if this lock (only) was
      OAF9 3212      ; responsible for the group grant mode. If no, then we can't
      OAF9 3213      ; grant any more locks because the group grant mode won't change.
      OAF9 3214
      OAF9 3215      ASSUME RSB$$_WTQFL EQ RSB$$_CVTQFL+8
      OAF9 3216
      53 38 A8 D0 OAF9 3217      MOVL RSB$$_CSID(R8),R3      ; Is resource managed by another system?
      09 13 OAFD 3218      BEQL 8$      ; No
      00000000 GF 16 OAFF 3219      JSB G^CNX$ALLOC_CDRP      ; No, get one (and convert CSID to CSB)
      BA 50 E9 OB05 3220      BLBC R0,NOCDRP      ; None available or CSID convert error
      35 A6 91 OB08 3221 8$:      CMPB LKB$$_GRMODE(R6),-      ; Is lock mode PW or higher?
      04      OB0B 3222      #LCK$$_PWMODE
      20 1F OB0C 3223      BLSSU 12$      ; No, skip value block processing
      59 D5 OB0E 3224      TSTL R9      ; Value block specified?
      10 13 OB10 3225      BEQL 10$      ; No
      28 A8 69 7D OB12 3226      MOVQ (R9),RSB$$_VALBLK(R8)      ; Yes, copy caller's value block to RSB
      30 A8 08 A9 7D OB16 3227      MOVQ 8(R9),RSB$$_VALBLK+8(R8)
      3C A8 D6 OB1B 3228      INCL RSB$$_VALSEQNUM(R8)      ; Increment value block sequence number
      02 AA OB1E 3229      BICW #RSB$$_VALINVL,-      ; Validate value block
      0E A8      OB20 3230      RSB$$_STATUS(R8)
      54 04 B3 OB22 3231 10$:      BITW #LCK$$_INVVALBLK,R4      ; Should value block be invalidated?
      07 13 OB25 3232      BEQL 12$      ; No
      02 A8 OB27 3233      BISW #RSB$$_VALINVL,-      ; Yes, invalidate value block
      0E A8      OB29 3234      RSB$$_STATUS(R8)
      3C A8 D6 OB2B 3235      INCL RSB$$_VALSEQNUM(R8)      ; Increment value block sequence number
      20 A6 D5 OB2E 3236 12$:      TSTL LKB$$_BLKASTADR(R6)      ; Blocking AST address specified?
      03 13 OB31 3237      BEQL 15$      ; No
      42 A8 B7 OB33 3238      DECW RSB$$_BLKASTCNT(R8)      ; Decr. blocking AST count
      53 D5 OB36 3239      OB36 3239
      94 12 OB38 3241      TSTL R3      ; Resource managed remotely?
      50 38 A6 OF OB3A 3242      BNEQ SNDDEQ GRNT      ; Yes
      20 12 OB3E 3243      REMQUE LKB$$_SQFL(R6),R0      ; Remove lock from granted queue
      50 18 A8 DE OB40 3244      BNEQ 45$      ; Branch if queue not empty
      50 60 D1 OB44 3245      MOVAL RSB$$_CVTQFL(R8),R0      ; Get address of conversion queue
      1E 12 OB47 3246      CMPL (R0),R0      ; Is conversion queue empty?
      50 08 C0 OB49 3247      BNEQ 50$      ; No
      50 60 D1 OB4C 3248      ADDL #8,R0      ; Yes, get address of wait queue
      05 12 OB4F 3249      CMPL (R0),R0      ; Is wait queue empty?
      OB51 3250      BNEQ 35$      ; No, try granting waiters
      OB51 3251      ; All queues are empty. Deallocate RSB (as long as it's not
      OB51 3252      ; a directory entry).
      00DD 30 OB51 3253
      22 11 OB54 3254      BSBW LCK$$_DEALLOC_RSB
      OB56 3255      BRB 60$      ; Finish up
      OB56 3256
      OB56 3257 35$:      ; Try granting waiting locks
      OB56 3258
      OB56 3259      ASSUME RSB$$_CGMODE EQ RSB$$_GGMODE+1
      OB56 3260
      0C A8 B4 OB56 3261      CLRW RSB$$_GGMODE(R8)      ; Clear group and conversion grant mode
      55 D4 O559 3262      CLRL R5      ; Clear group grant mode in R5
      OB5B 3263
      FDBD 30 OB5B 3264 40$:      BSBW LCK$$_GRANTWTRS      ; Try granting waiters
      18 11 OB5E 3265      BRB 60$
      OB60 3266
      OB60 3267 45$:      ; Determine if the lock dequeued was equal to the conversion
```

				OB60	3268	; grant mode. If not, then no new locks can be granted.
				OB60	3269	
				OB60	3270	
				OB63	3271	CMPB LKBSB_GRMODE(R6),- ; Is the granted mode of this lock
				OB65	3272	RSBSB_CGMODE(R8) ; equal to the conversion grant mode?
				OB67	3273	BNEQ 60\$; No, don't bother going further
				OB67	3274	50\$: ; Either we dequeued a lock equal to the conversion grant mode
				OB67	3275	; or we dequeued the head of the conversion queue. Either way,
				OB67	3276	; we must recompute the group grant mode.
				OB67	3277	
				OB67	3278	BSBW LCK\$COMP_GGMODE ; New group grant mode in R5
				OB6A	3279	MOVB R5,RSBSB_GGMODE(R8) ; Store in RSB
				OB6E	3280	MOVB R5,RSBSB_CGMODE(R8) ; Also store conversion grant mode
				OB72	3281	BSBW LCK\$GRANTCVTS ; Try granting conversions and waiters
				OB75	3282	BSBW LCK\$CHECK_RSB ; Deallocate RSB, if necessary
				OB78	3283	
				OB78	3284	60\$: ; Now finish cleaning up the lock we originally dequeued.
				OB78	3285	; First, decrement parent LKB's sub LKB reference count.
				OB78	3286	
				OB78	3287	MOVBL LKB\$L_PARENT(R6),R0 ; Get parent LKB address
				OB7C	3288	BEQL 65\$; No parent
				OB7E	3289	DECW LKB\$W_REFCNT(R0) ; Decrement parent's sub LKB ref. count
				OB81	3290	BLSS 75\$; Ref. count went negative
				OB83	3291	
				OB83	3292	65\$: ; Deallocate lock id
				OB83	3293	
				OB83	3294	MOVZWL LKB\$L_LKID(R6),R0 ; Get lock id index
				OB87	3295	MOVL G^LCK\$GL_IDTBL,R1 ; *** Combine with next instr.
				OB8E	3296	MOVAL (R1)[R0],R1 ; Point to table entry
				OB92	3297	MOVW G^LCK\$GL_NXTID,(R1) ; Store next id in this id's slot
				OB99	3298	ADDW3 #1,LKB\$L_LKID+2(R6),2(R1) ; Incr. and store sequence number
				OB9F	3299	BVC 70\$; Didn't overflow so a system address
				OBA1	3300	MOVW #1,2(R1) ; Overflowed - restart seq. number at 1
				OBA5	3301	MOVL R0,G^LCK\$GL_NXTID ; This id becomes the next one
				OBA5	3302	
				OBA5	3303	; If this lock is a master copy or system owned, then skip
				OBA5	3304	; following "process" code and just deallocate LKB. Both of these
				OBA5	3305	; conditions have a zero PID field in the LKB.
				OBA5	3306	
				OBA5	3307	TSTL LKB\$L_PID(R6) ; Is it either?
				OBAF	3308	BEQL 87\$; Yes
				OBB1	3309	
				OBB1	3310	; Remove LKB from owner's lock queue. If the lock was not
				OBB1	3311	; granted yet, then complete the request (queue an AST and
				OBB1	3312	; set event flag) with the status in R7.
				OBB1	3313	
				OBB1	3314	ASSUME LKB\$K_GRANTED EQ 1
				OBB1	3315	ASSUME LKB\$K_CONVERT EQ 0
				OBB1	3316	ASSUME LKB\$K_WAITING EQ -1
				OBB1	3317	
				OBB1	3318	REMQUE LKB\$L_OWNOFL(R6),R0
				OBB5	3319	TSTB LKB\$B_STATE(R6) ; Is the lock granted?
				OBB8	3320	BGTR 80\$; Yes
				OBB8A	3321	MOVL R7,LKB\$L_LKST1(R6) ; No, store specified status
				OBBE	3322	BNEQ 73\$; Had one
				OBC0	3323	MOVZWL S^#SS\$ ABORT,- ; Use default status instead
				OBC2	3324	LKB\$L_LKST1(R6)

```

      20 8A OBC4 3325 73$: BICB #LKB$M NODELETE,- ; Clear nodelete bit
OB A6    OBC6 3326      LKB$B_RMOD(R6)
FAE3    30 OBC8 3327      BSBW QUEUE_AST ; Queue AST and set event flag;
      OBCB 3328      ; returns status on R0. Kernel AST
      OBCB 3329      ; routine will delete LKB.
      05 OBCB 3330      RSB
      OBCB 3331
      OBCB 3332 75$: BUG_CHECK LKBREFNEG,FATAL
      OBD0 3333
      OBD0 3334 80$: ; Increment the enqueue count and deallocate the LKB as long
      OBD0 3335      ; as it's not queued to deliver an AST. If it is queued, then
      OBD0 3336      ; if it's queued to deliver a completion AST then let kernel AST
      OBD0 3337      ; routine delete the LKB. If it's only queued for a blocking
      OBD0 3338      ; AST then remove it from the AST queue.
      OBD0 3339
      2A A6 03 B3 OBD0 3340 BITW #LKB$M DCPLAST!- ; Is the LKB queued for AST delivery?
      OBD4 3341      LKB$M_DBLKAST,LKB$W_STATUS(R6)
      18 13 OBD4 3342      BEQL 85$ ; No
      20 8A OBD6 3343      BICB #LKB$M NODELETE,- ; Yes, clear nodelete bit
OB A6    OBD8 3344      LKB$B_RMOD(R6)
      AA OBD8 3345      BICW #LKB$M_DBLKAST- ; Clear deliver blocking AST bit
      OBD8 3346      !LKB$M_CVTTOSYS,- ; and convert to system owned bit
      OBD8 3347      LKB$W_STATUS(R6)
      2A A6 0102 8F OBD8 3348      BBS #LKB$V DCPLAST,- ; Branch if queued for a completion AST
      00 OBE0 3349      LKB$W_STATUS(R6),90$ ; (will be deall. when AST is delivered)
      2F 2A A6 OBE2 3349      MOVL R6,R5 ; Blocking AST only - remove from
      55 56 D0 OBE5 3350      JSB G^SCH$REMOVACB ; AST queue
      00000000'GF 16 OBE8 3351
      OBE8 3352
      OBE8 3353 85$: ; Deallocate LKB and increment enqueue quota (if it was charged)
      OBE8 3354
      OBE8 3355
      20 B3 OBE8 3356 BITW #LKB$M NOQUOTA,- ; Was enqueue quota charged?
      2A A6 OBF0 3357      LKB$W_STATUS(R6)
      17 12 OBF2 3357      BNEQ 87$ ; No
      54 0C A6 3C OBF4 3358      MOVZWL LKB$L_PID(R6),R4 ; Get process index
      50 00000000'GF D0 OBF8 3359      MOVL G^SCH$GL_PCBVEC,R0 ; *** Combine this and next inst. when
      OBF8 3360      ; PIC code is no longer needed ***
      54 6044 D0 OBF8 3361      MOVL (R0)[R4],R4 ; Convert to PCB address
      50 0080 C4 D0 OC03 3362      MOVL PCB$L_JIB(R4),R0 ; Get address of JIB
      4C A0 B6 OC08 3363      INCW JIB$W_ENQCNT(R0) ; Increment enqueue count
      50 56 D0 OC08 3364 87$: MOVL R6,R0 ; Address of LKB
      00000000'GF 16 OC0E 3365      JSB G^EXE$DEANONPAGED ; Deallocate it
      OC14 3366
      50 01 3C OC14 3367 90$: MOVZWL S^#SS$_NORMAL,R0
      05 OC17 3368      RSB
      OC18 3369
      OC18 3370
      OC18 3371      .DSABL LSB
```

PSE

SAB
LOC
YSE

Pha

Ini
Com
Pas
Sym
Pas
Sym
Pse
Cro
Ass

The

125

The

377

30

Mac

-S2

-S2

TOT

144

The

MAC

```
OC18 3373 .SBTTL LCK$CHECK_RSB - Deallocate RSB if necessary
OC18 3374
OC18 3375 :++
OC18 3376 : FUNCTIONAL DESCRIPTION:
OC18 3377 :
OC18 3378 : This routine checks to see if all queues on a resource
OC18 3379 : are empty. If they are, the resource can be deleted
OC18 3380 : as long as it's not needed to act as a directory entry.
OC18 3381 :
OC18 3382 : CALLING SEQUENCE:
OC18 3383 :
OC18 3384 : BSBW LCK$CHECK_RSB
OC18 3385 : BSBW LCK$DEALLOC_RSB is an entry point to use if all three queues
OC18 3386 : are known to be empty.
OC18 3387 : IPL must be at IPL$_SYNCH
OC18 3388 :
OC18 3389 : INPUT PARAMETERS:
OC18 3390 :
OC18 3391 : R8 Address of RSB
OC18 3392 :
OC18 3393 : OUTPUT PARAMETERS:
OC18 3394 :
OC18 3395 : None
OC18 3396 :
OC18 3397 : SIDE EFFECTS:
OC18 3398 :
OC18 3399 : R0 - R5 are destroyed
OC18 3400 :
OC18 3401 : NOTES:
OC18 3402 :
OC18 3403 : If all queues are empty and the RSB is not a root RSB then it
OC18 3404 : can be deleted. If it is a root RSB then the situation is more
OC18 3405 : complicated as the RSB may still be needed to act as a directory
OC18 3406 : entry. The following table summarizes what action must be taken.
OC18 3407 :
OC18 3408 : This is a directory entry
OC18 3409 : (RSB$_DIRENTRY = 1)
OC18 3410 :
OC18 3411 :
OC18 3412 :
OC18 3413 :
OC18 3414 : This system
OC18 3415 : is managing
OC18 3416 : the resource
OC18 3417 : (RSB$_CSID =
OC18 3418 : 0)
OC18 3419 :
OC18 3420 :
OC18 3421 : --
OC18 3422 :
OC18 3423 :
OC18 3424 : .ENABL LSB
OC18 3425 :
OC18 3426 : ASSUME RSB$_CMTQFL EQ RSB$_GRQFL+8
OC18 3427 : ASSUME RSB$_WTQFL EQ RSB$_CMTQFL+8
OC18 3428 :
OC18 3429 LCK$CHECK_RSB::
```

```
50 10 A8 DE OC18 3430 MOVAL RSB$L_GRQFL(R8),R0 ; Get address of granted queue
50 60 D1 OC1C 3431 CMPL (R0),R0 ; Is granted queue empty?
50 58 12 OC1F 3432 BNEQ 55$ ; No
50 08 C0 OC21 3433 ADDL #8,R0 ; Yes, get address of conversion queue
50 60 D1 OC24 3434 CMPL (R0),R0 ; Is conversion queue empty?
50 50 12 OC27 3435 BNEQ 55$ ; No
50 08 C0 OC29 3436 ADDL #8,R0 ; Yes, get address of wait queue
50 60 D1 OC2C 3437 CMPL (R0),R0 ; Is wait queue empty?
50 48 12 OC2F 3438 BNEQ 55$ ; No
OC31 3439
OC31 3440 LCK$DEALLOC_RSB::
OC31 3441 ; All queues are empty. Delete RSB and/or send message to
OC31 3442 ; directory system as appropriate (see above table).
OC31 3443
OC31 3444 ASSUME RSB$L_HSHCHN EQ 0
OC31 3445 ASSUME RSB$L_HSHCHNBK EQ RSB$L_HSHCHN+4
OC31 3446 ASSUME RSB$M_DIRENTRY EQ 1
OC31 3447
40 A8 B5 OC31 3448 TSTW RSB$W_REFCNT(R8) ; Verify there are no sub RSB's
44 12 OC34 3449 BNEQ 60$ ; There are
50 48 A8 D0 OC36 3450 MOVL RSB$L_PARENT(R8),R0 ; Get parent RSB address
23 12 OC3A 3451 BNEQ 25$ ; There is a parent
OC3C 3452
07 0E A8 E9 OC3C 3453 BLBC RSB$W_STATUS(R8),15$ ; Branch if this is not a dir. entry
38 A8 D5 OC40 3454 TSTL RSB$L_CSID(R8) ; Is this system managing this resource?
1F 13 OC43 3455 BEQL 35$ ; Yes - resource can be deleted
32 11 OC45 3456 BRB 55$ ; This is a directory entry; don't delete
38 A8 D5 OC47 3457 15$: TSTL RSB$L_CSID(R8) ; Is this system managing the resource?
18 12 OC4A 3458 BNEQ 35$ ; No - Just delete resource
OC4C 3459
OC4C 3460 ; Have to send a remove directory entry message to directory system
OC4C 3461
50 68 7D OC4C 3462 MOVQ RSB$L_HSHCHN(R8),R0 ; Get hash chain pointers in R0 and R1
61 50 D0 OC4F 3463 MOVL R0,RSB$L_HSHCHN(R1) ; Store next pointer in previous RSB or
OC52 3464 ; hash table
OC52 3465 BEQL 20$ ; Branch if no next one
04 A0 51 D0 OC54 3466 MOVL R1,RSB$L_HSHCHNBK(R0) ; Store previous pointer in next one
00000000'GF 16 OC58 3467 20$: JSB G^LCK$END_RMVDIR ; Send remove directory entry message
05 OC5E 3468 RSB
OC5F 3469
40 A0 B7 OC5F 3470 25$: DECW RSB$W_REFCNT(R0) ; Decrement parent's sub RSB ref. count
1A 19 OC62 3471 BLSS 70$ ; Ref. count went negative
50 68 7D OC64 3472 35$: MOVQ RSB$L_HSHCHN(R8),R0 ; Get hash chain pointers in R0 and R1
61 50 D0 OC67 3473 MOVL R0,RSB$L_HSHCHN(R1) ; Store next pointer in previous RSB or
OC6A 3474 ; hash table
OC6A 3475 BEQL 45$ ; Branch if no next one
04 A0 51 D0 OC6C 3476 MOVL R1,RSB$L_HSHCHNBK(R0) ; Store previous pointer in next one
50 58 D0 OC70 3477 45$: MOVL R8,R0
00000000'GF 16 OC73 3478 JSB G^EXE$DEANONPAGED ; Deallocate RSB
05 OC79 3479 55$: RSB
OC7A 3480
OC7A 3481 60$: BUG_CHECK RSBREFNZRO,FATAL
OC7E 3482 70$: BUG_CHECK RSBREFNEG,FATAL
OC82 3483
OC82 3484 .DSABL LSB
```

```
0C82 3486 .SBTTL STALL_REQ - Stall request during failover
0C82 3487
0C82 3488 :++
0C82 3489 : FUNCTIONAL DESCRIPTION:
0C82 3490 :
0C82 3491 : This routine stalls requests during failover by putting the
0C82 3492 : process into MWAIT state waiting for resource RSNS_CLUSTERAN.
0C82 3493 : It is assumed that the request has been backed up and all cleanup
0C82 3494 : has been performed as this routine backs up the service and
0C82 3495 : waits in the caller's mode.
0C82 3496 :
0C82 3497 : The alternate entry point WAIT_FOR_POOL operates in the same
0C82 3498 : way but waits for non-paged pool as opposed to a cluster transition.
0C82 3499 :
0C82 3500 : The alternate entry point LCK$CHECK_STALL includes a test to
0C82 3501 : determine if we should stall.
0C82 3502 :
0C82 3503 : CALLING SEQUENCE:
0C82 3504 :
0C82 3505 : BRW STALL_REQ
0C82 3506 : BRW WAIT_FOR_POOL
0C82 3507 :
0C82 3508 : NOTE: These routines do not return to the caller. Rather they back
0C82 3509 : up the service and wait in the mode of the caller. When
0C82 3510 : the resource becomes available, the service is re-executed.
0C82 3511 :
0C82 3512 : JSB LCK$CHECK_STALL (Either returns to caller or backs up
0C82 3513 : system service call)
0C82 3514 : IPL must be at IPL$SYNCH
0C82 3515 :
0C82 3516 : INPUT PARAMETERS:
0C82 3517 :
0C82 3518 : None
0C82 3519 :
0C82 3520 : IMPLICIT INPUTS:
0C82 3521 :
0C82 3522 : It is assumed that these routines are being called from the context
0C82 3523 : of a system service and that FP has not been tinkered with.
0C82 3524 :
0C82 3525 : OUTPUT PARAMETERS:
0C82 3526 :
0C82 3527 : None
0C82 3528 :
0C82 3529 : SIDE EFFECTS:
0C82 3530 :
0C82 3531 : The service is backed out
0C82 3532 : --
0C82 3533 :
0C82 3534 : WAIT_FOR_POOL:
0C82 3535 : MOVL #RSNS_NPDYNMEM,R0 ; Set resource to wait for
0C82 3536 : BRB WAIT_COM
0C82 3537 : LCK$CHECK_STALL:
0C82 3538 : TSTB G^LCK$GB_STALLREQS ; Are we stalling all requests?
0C82 3539 : BLSS STALL_REQ ; Yes
0C82 3540 : RSB
0C82 3541 : STALL_REQ:
0C82 3542 : MOVL #RSNS_CLUSTERAN,R0 ; Set resource to wait for
```

50 03 D0 0C82 3535
0C 11 0C85 3536
00000000'GF 95 0C87 3537
01 19 0C8D 3538
05 0C8F 3539
50 0E D0 0C90 3540
0C90 3541
0C90 3542

```

54 00000000'GF  D0 0C93 3543 WAIT_COM:
      SE 5D  D0 0C93 3544      MOVL G^SCH$GL_CURPCB,R4      ; Set our PCB address
      SC 08 AE 7D 0C9A 3545      MOVL FP,SP      ; Trim stack back to start of frame
      SE 00'  C0 0C9D 3546      MOVQ 8(SP),AP      ; Restore pre-call AP and FP
      6E 04  C0 0CA1 3547      ADDL S^#EX$C_CMSTKSZ,SP      ; Clean call frame off stack
00000000'GF  C2 0CA4 3548      SUBL #4,(SP)      ; Back up saved PC to point to CHMK
      17 0CA7 3549      JMP G^SCH$RWAIT      ; Wait

```

```
OCAD 3551 .SBTTL LCK$EXTEND_IDTBL - Extend lock id. table
OCAD 3552
OCAD 3553
OCAD 3554 :++
OCAD 3555 : FUNCTIONAL DESCRIPTION:
OCAD 3556 : This routine extends the lock id. table if it hasn't already
OCAD 3557 : reached it's maximum size.
OCAD 3558
OCAD 3559 : CALLING SEQUENCE:
OCAD 3560 : BSBW LCK$EXTEND_IDTBLW - To be called when in process context.
OCAD 3561 : If non-paged pool is not available
OCAD 3562 : the system service will be backed out
OCAD 3563 : and the caller will wait.
OCAD 3564 : BSBW LCK$EXTEND_IDTBL - To be called when in fork context.
OCAD 3565 : This entry point will not wait for pool
OCAD 3566 : and will instead return an error to the
OCAD 3567 : caller.
OCAD 3568 : IPL must be at IPL$_SYNCH
OCAD 3569
OCAD 3570 : INPUT PARAMETERS:
OCAD 3571 : R0 Address of cleanup routine or 0 (LCK$EXTEND_IDTBLW only)
OCAD 3572
OCAD 3573 : IMPLICIT INPUTS:
OCAD 3574 : Various lock manager memory cells and SYSGEN parameters
OCAD 3575 : (LCK$GL_MAXID, LCK$GL_IDTBL$IZ, etc.)
OCAD 3576
OCAD 3577 : OUTPUT PARAMETERS:
OCAD 3578 : R0 Completion code
OCAD 3579
OCAD 3580 : COMPLETION CODES:
OCAD 3581 : $$$_INSFMEM Insufficient non-paged pool
OCAD 3582 : $$$_NOLOCKID Table has already been expanded to the maximum
OCAD 3583
OCAD 3584 : SIDE EFFECTS:
OCAD 3585 : R1 - R4 are destroyed
OCAD 3586 :--
OCAD 3587
OCAD 3588 .ENABL LSB
OCAD 3589
OCAD 3590 50 0E12 8F 3C OCAD 3596 5$: MOVZWL #$$$_NOLOCKID,R0 ; Indicate error
OCAD 3597 10$: RSB
OCAD 3598
OCAD 3599 LCK$EXTEND_IDTBL::
OCAD 3600 MOVAB G^EXESALONONPAGED,R2 ; Address of allocate routine
OCAD 3601 BRB 15$
OCAD 3602
OCAD 3603 LCK$EXTEND_IDTBLW::
OCAD 3604 MOVAB G^EXESALONPAGWAITS,R2 ; Address of allocate routine
OCAD 3605
OCAD 3606 15$: ; Check that we haven't already exceeded the maximum table size
OCAD 3607
```



```
51 00000000'GF D0 OCC3 3608      MOVL  G^LCK$GL MAXID,R1      ; Get current largest lock id.
00000000'GF 51 D1 OCCA 3609      CMPL  R1,G^LCK$GL_IDTBLMAX    ; Have we reached the upper limit?
54 51 01 DA 1E OCD1 3610      BGEQU  SS                      ; Yes, return SS$ NOLOCKID
                                C1 OCD3 3611      ADDL3  #1,R1,R4      ; Incr. and save for later
                                OCD7 3612
                                OCD7 3613      ; Compute new table size and allocate it from pool.
                                OCD7 3614
51 00000000'GF C0 OCD7 3615      ADDL  G^LCK$GL_IDTBLSIZ,R1    ; Add another increment to table size
51 04 C4 OCDE 3616      MULL  #4,R1                      ; Convert to size in bytes
51 0C C0 OCE1 3617      ADDL  #12,R1                     ; Add header
62 16 OCE4 3618      JSB  (R2)                          ; Allocate it
C9 50 E9 OCE6 3619      BLBC  R0,10$                     ; Insuff. memory
                                OCE9 3620
                                OCE9 3621      ; Copy old table into new and deallocate old table. Registers contain:
                                OCE9 3622      R1      Allocated size of new table
                                OCE9 3623      R2      Address of new table
                                OCE9 3624      R4      Old maximum lock id. + 1
                                OCE9 3625
55 00000000'GF 36 BB OCE9 3626      PUSHR  #^M<R1,R2,R4,R5>    ; Save regs.
50 55 OC C3 OCEB 3627      SUBL3  #12,G^LCK$GL_IDTBL,R5    ; Get starting address of old table
53 65 FE 8F 78 OCF6 3628      MOVL  R5,R0                ; Save in R0
82 85 DO OCF3 3629      ASHL  #2,(R5),R3                ; Get size of old table in longwords
FA 53 F5 OCFB 3630      MOVL  (R5)+(R2)+                ; Move old table entry to new table
51 60 DO OCFE 3631      SOBGTR R3,20$                    ; Repeat
00000000'GF 16 DO OD01 3632      MOVL  (R0),R1            ; Get size of old table
36 BA OD04 3633      JSB  G^EX$DEANONPGDSIZ              ; Deallocate old table
                                OD0A 3634      POPR  #^M<R1,R2,R4,R5> ; Restore regs.
                                OD0C 3635
                                OD0C 3636      ; Set up header of new table and pointers to it. Registers contain:
                                OD0C 3637      R1      Size of table
                                OD0C 3638      R2      Address of table
                                OD0C 3639      R4      Old maximum lock id. + 1
                                OD0C 3640
62 51 D0 OD0C 3641      MOVL  R1,(R2)                    ; Store size in first longword
08 A2 B4 OD0F 3642      CLRW  8(R2)                      ; Clear old size in word size field
02 A2 B5 OD12 3643      TSTW  2(R2)                      ; Will size fit in a word?
04 12 OD15 3644      BNEQ  30$                            ; No
08 A2 51 B0 OD17 3645      MOVW  R1,8(R2)                ; Yes, store it in normal place
00000000'GF OC A2 9E OD1B 3646      MOVAB 12(R2),G^LCK$GL_IDTBL ; Store pointer to table
00000000'GF 54 D0 OD23 3647      MOVL  R4,G^LCK$GL_NXTID    ; Store next lock id. to allocate
51 10 C2 OD2A 3648      SUBL  #16,R1                    ; Compute new max. lock id. (# of
51 04 C6 OD2D 3649      DIVL  #4,R1                      ; entries in table - 1)
50 FFFF 8F 3C OD30 3650      MOVZWL #^XFFFF,R0           ; Load useful constant (65535)
50 51 D1 OD35 3651      CMPL  R1,R0                      ; Make sure we don't exceed 65535
03 1B OD38 3652      BLEQU  40$                            ; We're okay
51 50 D0 OD3A 3653      MOVL  R0,R1                      ; Set number of entries to exactly 65535
00000000'GF 51 D0 OD3D 3654      MOVL  R1,G^LCK$GL MAXID    ; Set maximum lock id.
53 OC A244 DE OD44 3655      MOVAL 12(R2)[R4],R3          ; Point to first new entry
54 D6 OD49 3656      INCL  R4                            ; Increment next lock id.
50 D6 OD4B 3657      INCL  R0                            ; Change constant to ^X10000
                                OD4D 3658
                                OD4D 3659      ; Now initialize new section of table by storing linked list
                                OD4D 3660      ; of lock id. indices and sequence numbers. Registers contain:
                                OD4D 3661      R0      Constant ^X10000
                                OD4D 3662      R1      Maximum lock id.
                                OD4D 3663      R3      Address of first new entry
                                OD4D 3664      R4      Next lock id. to store in table
```

54	50	C8	OD4D	3665			
83	54	D0	OD4D	3666			
	54	B6	OD50	3667	50\$:		
51	54	B1	OD53	3668			
	F6	15	OD55	3669			
63	50	D0	OD58	3670			
50	01	D0	OD5A	3671			
		D0	OD5D	3672			
		05	OD60	3673			
			OD61	3674			
			OD61	3675			

BISL	R0,R4	:	Logically OR seq. num. with next id.
MOVL	R4,(R3)+	:	Store next table entry
INCL	R4	:	Incr. next id.
CMPW	R4,R1	:	Compare with max. id
BLEQ	50\$	:	Repeat
MOVL	R0,(R3)	:	Store last entry
MOVL	S^#SS\$ _NORMAL,R0	:	Indicate success
RSB			
.DSABL	LSB		

```
OD61 3677 .SBTTL FREE_LKB - Free LKB (from AST queue)
OD61 3678
OD61 3679 :++
OD61 3680 : FUNCTIONAL DESCRIPTION:
OD61 3681 :
OD61 3682 : This routine is called to free an LKB that is currently
OD61 3683 : queued as an ACB. This should happen rarely, but when it does
OD61 3684 : an ACB is allocated from pool, the ACB portion of the LKB is copied
OD61 3685 : into the new ACB and the two ACBs are swapped on the AST queue.
OD61 3686 : This frees up the LKB for another use (such as a lock conversion).
OD61 3687 :
OD61 3688 : CALLING SEQUENCE:
OD61 3689 :
OD61 3690 : BSBW FREE_LKB
OD61 3691 : (Note: IPL must be at IPL$_ASTDEL or lower)
OD61 3692 :
OD61 3693 : INPUT PARAMETERS:
OD61 3694 :
OD61 3695 : R4 Address of PCB
OD61 3696 : R6 Address of LKB
OD61 3697 :
OD61 3698 : OUTPUT PARAMETERS:
OD61 3699 :
OD61 3700 : R0 Completion Code (returned to ERROR_EXIT - not caller)
OD61 3701 :
OD61 3702 : COMPLETION CODES:
OD61 3703 :
OD61 3704 : $$$_INSFMEM Insufficient memory
OD61 3705 : $$$_EXASTLM Exceeded AST quota
OD61 3706 :
OD61 3707 : SIDE EFFECTS:
OD61 3708 :
OD61 3709 : The LKB is removed from the AST queue. Note that all registers
OD61 3710 : (including R0 and R1) must be preserved.
OD61 3711 :
OD61 3712 : NOTES:
OD61 3713 :
OD61 3714 : This code makes two assumptions:
OD61 3715 :
OD61 3716 : 1) That the LKB must be queued for a regular AST
OD61 3717 : (as opposed to just a special kernel AST). This
OD61 3718 : is why AST quota is always deducted, not conditionally
OD61 3719 : on whether an AST address was specified.
OD61 3720 :
OD61 3721 : 2) That the LKBSM_DCPLAST and LKBSM_DBLKAST bits cannot
OD61 3722 : become clear while we are at IPL=0. Otherwise, it
OD61 3723 : is necessary to verify that the LKB is still in use
OD61 3724 : after the ACB is allocated from pool. This assumption
OD61 3725 : is due to the fact that the AST must either be for
OD61 3726 : an outer mode or if for kernel mode then kernel mode
OD61 3727 : ASTs must be disabled.
OD61 3728 : --
OD61 3729 :
OD61 3730 : .IF NDF LOADSW
00000251 3731 : PSECT Y$EXEPAGED
0251 3732 : .ENDC
0251 3733
```

```
0251 3734 FREE_LKB:
      3F 8B 0251 3735 PUSHR #*M<R0,R1,R2,R3,R4,R5>
    38 A4 B5 0253 3736 TSTW PCBSW_ASTCNT(R4) ; Test for enough AST quota
      50 15 0256 3737 BLEQ 80$ ; Error - not enough
    51 34 3C 0258 3738 MOVZWL #LKBSK_ACBLEN,R1 ; Size of ACB to allocate
00000000'GF 16 025B 3739 JSB G^EXES$ALLOCBUF ; Allocate ACB
      49 50 E9 0261 3740 BLBC R0,90$ ; Error - insuff. memory
    OA A2 02 90 0264 3741 MOVB #DYN$C_ACB,ACBSB_TYPE(R2) ; Store data structure type
      38 A4 B7 0268 3742 DECW PCBSW_ASTCNT(R4) ; Decrement AST quota
      026B 3743 10$: SETIPL 95$ ; Raise to IPL$_SYNCH and
      0272 3744 ; lock pages in memory
      0272 3745
      0272 3746 ; If CVTTOSYS bit is set, then this LKB should be converted
      0272 3747 ; to system owned. Do it now instead of in LOCK_KAST.
      0272 3748
      06 2A 08 E1 0272 3749 BBC #LKBSV_CVTTOSYS, - ; Branch if CVTTOSYS is clear
    000001DB'GF 16 0274 3750 LKBSW_STATUS(R6),20$
      0277 3751 JSB G^CVT-TO SYS_INT ; Convert to system owned
      AA 027D 3752 20$: BICW #LKBSM_DBLKAST- ; Clear deliver blocking AST bit
      027E 3753 ; !LKBSM_CVTTOSYS,- ; and convert to system owned bit
      027E 3754 LKBSW_STATUS(R6)
      2A A6 0102 8F 0283 3755 PUSHL R2 ; Save ACB address
      62 66 34 28 0285 3756 MOVCL #LKBSK_ACBLEN,(R6),(R2) ; Copy ACB portion of LKB
      52 8ED0 0289 3757 POPL R2 ; Restore ACB address
    OB A2 40 8F 88 028C 3758 BISB #ACBSM_QUOTA,ACBSB_RMOD(R2) ; Set quota accounting flag
      OB A2 20 8A 0291 3759 BICB #ACBSM_NODELETE,ACBSB_RMOD(R2) ; Clear nodelete flag
      2A A2 20 A8 0295 3760 BISW #LKBSM_NOQUOTA,- ; Set NOQUOTA bit so that
      55 56 D0 0299 3761 LKBSW_STATUS(R2) ; enqueue quota is not credited
00000000'GF 16 029C 3762 MOVL R6,R5 ; Address of LKB
      3F BA 02A2 3763 JSB G^SCH$SWAPACBS ; Swap ACBs
      02A4 3764 POPR #*M<R0,R1,R2,R3,R4,R5>
      05 02A7 3765 SETIPL #IPL$_ASTDEL ; Lower IPL
      02A8 3766 RSB
      50 2A04 8F 3C 02A8 3768 80$: MOVZWL #SS$ EXASTLM,R0
    00000543'EF 17 02AD 3769 90$: JMP ERROR_EXIT_R0
      02B3 3770
      00000008 02B3 3771 95$: .LONG IPL$ SYNCH ; End of locked down code
      02B7 3772 ASSUME .-10$ LE 512 ; Must be on adjoining pages
      02B7 3773
      02B7 3774
      02B7 3775 .END
```

SYSENQDEQ
Symbol table

- ENQUEUE/DEQUEUE SYSTEM SERVICES

M 4

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 82
(27)

```
ACBSB_RMOD      = 0000000B
ACBSB_TYPE      = 0000000A
ACBSM_NODELETE  = 00000020
ACBSM_QUOTA     = 00000040
ACCVIO          = 0000004C R      03
ACMODE          = 00000028
ASTADR          = 0000001C
ASTPRM          = 00000020
BLKAST          = 00000024
BUGS_DEQSUBLCKS ***** X      03
BUGS_LKBREFNEG ***** X      02
BUGS_LOCKMGRERR ***** X      02
BUGS_RSBREFNEG ***** X      02
BUGS_RSBREFNZRO ***** X      02
CAS_MEASURE     = 00000002
CALC_BLK_SUBR   = 0000089F R      02
CANCEL          = 00000A43 R      02
CANCELGRANT     = 00000A3D R      02
CLEANUP1        = 00000564 R      02
CLEANUP2        = 00000578 R      02
CLEANUP3        = 00000583 R      02
CLEANUP4        = 0000058E R      02
CNX$ALLOC_CDRP ***** X      02
CONVERSION      = 00000049 R      02
CVT_TO_PRC      = 00000208 R      02
CVT_TO_SYS      = 000001CB R      02
CVT_TO_SYS_INT  = 000001DB R      02
DEPTH_ERROR     = 00000504 R      02
DEQ_ACMODE      = 0000000C
DEQ_ALL         = 00000174 R      03
DEQ_CONVERT     = 00000AD7 R      02
DEQ_EXIT        = 0000015E R      03
DEQ_FLAGS       = 00000010
DEQ_GRANTED     = 00000AF9 R      02
DEQ_WAIT        = 00000AA9 R      02
DYN$C_ACB       = 00000002
DYN$C_LKB       = 00000035
DYN$C_RSB       = 00000036
EFN             = 00000004
ERROR_EXIT_R0   = 00000543 R      02
ERROR_EXIT_R11  = 00000540 R      02
EXES$ALOCBOF ***** X      03
EXES$ALONONPAGED ***** X      02
EXES$ALONPAGWAIT ***** X      03
EXES$ALONPAGWAITS ***** X      02
EXES$C_CMSTKSZ ***** X      02
EXES$DEANONPAGED ***** X      02
EXES$DEANONPGDSIZ ***** X      02
EXES$DEQ        = 00000111 RG     03
EXES$ENQ        = 0000000E RG     03
EXES$GL_ABSTIM ***** X      02
EXES$MAXACMODE ***** X      03
FLAGS           = 00000010
FREE_LKB        = 00000251 R      03
INVALID_STATE   = 00000A33 R      02
IOC$GL_SRPFL ***** X      03
IOC$GL_SRPSize ***** X      03
```

```
IPL$_ASTDEL     = 00000002
IPL$-SYNCH      = 00000008
JIB$Q_ENQCNT    = 0000004C
LCK$CANCEL_CVT  = 000009E8 RG     02
LCK$CHECK_RSB   = 00000C18 RG     02
LCK$CHECK-STALL = 00000C87 RG     02
LCK$COMPAT_TBL  = 00000000 RG     02
LCK$COMP_GMODE  = 000008AF RG     02
LCK$CVTNOTQED   = 00000194 RG     02
LCK$CVT_GRANTED = 0000016B RG     02
LCK$DEACLOC_RSB = 00000C31 RG     02
LCK$DEQLOCK     = 00000A85 RG     02
LCK$EXTEND_IDTBL = 00000CB3 RG     02
LCK$EXTEND-IDTBLW = 00000CBC RG     02
LCK$GB_HTBSHFT ***** X      02
LCK$GB_MAXDEPTH ***** X      02
LCK$GB-STALLREQS ***** X      02
LCK$GL_DIRVEC   ***** X      02
LCK$GL_HASHTBL ***** X      02
LCK$GL_IDTBL     ***** X      02
LCK$GL_IDTBLMAX ***** X      02
LCK$GL_IDTBLSIZ ***** X      02
LCK$GL_MAXID     ***** X      02
LCK$GL_NXTID     ***** X      02
LCK$GL_TIMEOUTQ ***** X      02
LCK$GL_WAITTIME ***** X      02
LCK$GRANTCVTS   = 000008D2 RG     02
LCK$GRANTWTRS   = 0000091B RG     02
LCK$GRANT_LOCK  = 0000060B RG     02
LCK$GRANT_LOCK_ALT = 00000610 RG     02
LCK$GRANT_REM    = 0000065D RG     02
LCK$HASH_SEARCH = 0000059C RG     02
LCK$K_CMODE     = 00000001
LCK$K_CWMODE    = 00000002
LCK$K_EXMODE    = 00000005
LCK$K_NLMODE    = 00000000
LCK$K_PMODE     = 00000003
LCK$K_PWMODE    = 00000004
LCK$LOCAL_CVT   = 0000010F RG     02
LCK$LOCAL_LOCK  = 000003D7 RG     02
LCK$M_CANCEL    = 00000002
LCK$M_CONVERT   = 00000002
LCK$M_CVTSYS    = 00000040
LCK$M_INVVALBLK = 00000004
LCK$M_NOQUEUE   = 00000004
LCK$M_NOQUOTA   = 00000020
LCK$M_PROTECT   = 00000100
LCK$M_RECOVER   = 00000080
LCK$M_SYNCSTS   = 00000008
LCK$M_SYSTEM    = 00000010
LCK$NORET_VALBLK = 000004FB RG     02
LCK$NOT_QDEUED  = 0000050D RG     02
LCK$QUEDECVT    = 000007AE RG     02
LCK$QUEUED_EXIT = 000004F8 RG     02
LCK$QUEUEWAIT   = 000007C8 RG     02
LCK$QUEUE_BLKAST = 00000878 RG     02
LCK$QUEUE_BLOCKAST = 0000082D RG     02
```

```

= 00000002
= 00000008
= 0000004C
000009E8 RG     02
00000C18 RG     02
00000C87 RG     02
00000000 RG     02
000008AF RG     02
00000194 RG     02
0000016B RG     02
00000C31 RG     02
00000A85 RG     02
00000CB3 RG     02
00000CBC RG     02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
***** X      02
000008D2 RG     02
0000091B RG     02
0000060B RG     02
00000610 RG     02
0000065D RG     02
0000059C RG     02
= 00000001
= 00000002
= 00000005
= 00000000
= 00000003
= 00000004
0000010F RG     02
000003D7 RG     02
= 00000002
= 00000002
= 00000040
= 00000004
= 00000004
= 00000020
= 00000100
= 00000080
= 00000008
= 00000010
000004FB RG     02
0000050D RG     02
000007AE RG     02
000004F8 RG     02
000007C8 RG     02
00000878 RG     02
0000082D RG     02
```

SYSENQDEQ
Symbol table

- ENQUEUE/DEQUEUE SYSTEM SERVICES N 4

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1Page 83
(27)

LCK\$QUEUE REM	000007CD	RG	02
LCK\$REGRANTLOCK	00000623	RG	02
LCK\$RET_VALBLK	000004D8	RG	02
LCK\$SND_BLKING	*****	X	02
LCK\$SND_CVTREQ	*****	X	02
LCK\$SND_DEQCV	*****	X	02
LCK\$SND_DEQGR	*****	X	02
LCK\$SND_DEQWT	*****	X	02
LCK\$SND_GRANTED	*****	X	02
LCK\$SND_LOCKREQ	*****	X	02
LCK\$SND_RMVDIR	*****	X	02
LCK\$SRCR_HSHBL	000005E4	RG	02
LCK\$SYNCCVT_TBL	00000006	RG	02
LCK\$SYNC_EXIT	000004C3	RG	02
LCK\$V_CONVERT	= 00000001		
LCK\$V_CVTSYS	= 00000006		
LCK\$V_DEQALL	= 00000000		
LCK\$V_NODLCKWT	= 00000009		
LCK\$V_NOQUEUE	= 00000002		
LCK\$V_NOQUOTA	= 00000005		
LCK\$V_PROTECT	= 00000008		
LCK\$V_RECOVER	= 00000007		
LCK\$V_VALBLK	= 00000000		
LKBSB_EFN	= 00000037		
LKBSB_GMODE	= 00000035		
LKBSB_RMOD	= 00000008		
LKBSB_RQMODE	= 00000034		
LKBSB_STATE	= 00000036		
LKBSB_TSLT	= 0000004E		
LKBSB_TYPE	= 0000000A		
LKBSK_ACBLN	= 00000034		
LKBSK_CONVERT	= 00000000		
LKBSK_GRANTED	= 00000001		
LKBSK_LENGTH	= 00000060		
LKBSK_WAITING	= FFFFFFFF		
LKBSL_AST	= 00000010		
LKBSL_ASTPRM	= 00000014		
LKBSL_ASTQFL	= 00000000		
LKBSL_BLKASTADR	= 00000020		
LKBSL_CPLASTADR	= 0000001C		
LKBSL_DUETIME	= 00000018		
LKBSL_KAST	= 00000018		
LKBSL_LKID	= 00000030		
LKBSL_LKSB	= 00000024		
LKBSL_LKST1	= 0000002C		
LKBSL_OLDASTPRM	= 00000058		
LKBSL_OLDBLKAST	= 0000005C		
LKBSL_OWNQFL	= 00000040		
LKBSL_PARENT	= 00000048		
LKBSL_PID	= 0000000C		
LKBSL_RSB	= 00000050		
LKBSL_SQFL	= 00000038		
LKBSM_ASYNC	= 00000004		
LKBSM_BLKASTQED	= 00000008		
LKBSM_CVTTOSYS	= 00000100		
LKBSM_DBLKAST	= 00000002		
LKBSM_DCPLAST	= 00000001		

LKBSM_KAST	= 00000080		
LKBSM_MSTCPY	= 00000010		
LKBSM_NODELETE	= 00000020		
LKBSM_NOQUOTA	= 00000020		
LKBSM_PKAST	= 00000010		
LKBSM_PROTECT	= 00000200		
LKBSM_TIMEOUTQ	= 00000040		
LKBSM_WASSYSOWN	= 00000080		
LKBSM_MODE	= 00000002		
LKBSV_ASYNC	= 00000002		
LKBSV_BLKASTQED	= 00000003		
LKBSV_CVTTOSYS	= 00000008		
LKBSV_DBLKAST	= 00000001		
LKBSV_DCPLAST	= 00000000		
LKBSV_MODE	= 00000000		
LKBSV_MSTCPY	= 00000004		
LKBSV_NODELETE	= 00000005		
LKBSV_NOQUOTA	= 00000005		
LKBSV_PROTECT	= 00000009		
LKBSV_TIMEOUTQ	= 00000006		
LKBSV_WASSYSOWN	= 00000007		
LKBSW_FLAGS	= 00000028		
LKBSW_REFCNT	= 0000004C		
LKBSW_SIZE	= 00000008		
LKBSW_STATUS	= 0000002A		
LKMODE	= 00000008		
LKSB	= 0000000C		
LOCKID	= 00000004		
LOCK_KAST	0000070A	R	02
NEW_LOCK	0000005C	R	03
NEW_RESOURCE	00000433	R	02
NOCGRP	00000AC2	R	02
OLD_RESOURCE	000003C1	R	02
PARTID	= 00000018		
PCBSL_JIB	= 00000080		
PCBSL_LOCKQBL	= 00000108		
PCBSL_LOCKQFL	= 00000104		
PCBSL_PID	= 00000060		
PCBSL_STS	= 00000024		
PCBSQ_PRIV	= 00000084		
PCBSV_RECOVER	= 0000001A		
PCBSW_ASTCNT	= 00000038		
PCBSW_GRP	= 000000BE		
PMSSGL_BLK_LOC	*****	X	02
PMSSGL_DEQ_LOC	*****	X	03
PMSSGL_ENQCVT_LOC	*****	X	02
PMSSGL_ENQNEW_LOC	*****	X	02
PMSSGL_ENQNOTQD	*****	X	02
PMSSGL_ENQWAIT	*****	X	02
POOL_MASK	= 0000000F		
PR\$ IPL	= 00000012		
PRIS_RESAVL	= 00000002		
PROT	= 0000002C		
PRVSV_SYSLCK	= 0000001E		
PSL\$C_EXEC	= 00000001		
PSL\$S_PVMOD	= 00000002		
PSL\$V_PVMOC	= 00000016		

SYSENQDEQ
Symbol table

- ENQUEUE/DEQUEUE SYSTEM SERVICES B 5

16-SEP-1984 02:02:16 VAX/VMS Macro V04-00
5-SEP-1984 03:52:48 [SYS.SRC]SYSENQDEQ.MAR;1

Page 84
(27)

SYSE
V04-

QUEUE_AST	000006AE	R	02	SS\$-INSFMEM	=	00000124	
QUEUE-BLKAST	0000087D	R	02	SS\$-IVBUFLN	=	0000034C	
QUEUE-COMMON	000007D8	R	02	SS\$-IVLOCKID	=	00002124	
REFCNT_ERROR	00000A37	R	02	SS\$-NOLOCKID	=	00000E12	
REM_LOCK	0000042D	R	02	SS\$-NOPRIV	=	00000024	
RESNAM	= 00000014			SS\$-NORMAL	=	00000001	
RSB\$B-CGMODE	= 0000000D			SS\$-NOSYSLCK	=	000028F4	
RSB\$B-DEPTH	= 0000000B			SS\$-NOTQUEUED	=	000009B8	
RSB\$B-GGMODE	= 0000000C			SS\$-PARNOTGRANT	=	00002134	
RSB\$B-RMOD	= 0000004E			SS\$-PARNOTSYS	=	0000225C	
RSB\$B-RSNLEN	= 0000004F			SS\$-RETRY	=	00000E32	
RSB\$B-TYPE	= 0000000A			SS\$-SUBLOCKS	=	0000212C	
RSB\$K-LENGTH	= 00000050			SS\$-SYNCH	=	00000689	
RSB\$K-MAXLEN	= 0000001F			SS\$-VALNOTVALID	=	000009F0	
RSB\$K-CSID	= 00000038			STALL REQ		00000C90	R 02
RSB\$K-CVTQBL	= 0000001C			VALBLR	=	00000008	
RSB\$K-CVTQFL	= 00000018			VERIFYLOCKID		00000968	R 02
RSB\$K-GRQFL	= 00000010			VERIFYPARLOCKID		0000096C	R 02
RSB\$K-HSHCHN	= 00000000			WAIT_COM		00000C93	R 02
RSB\$K-HSHCHNBK	= 00000004			WAIT_FOR_POOL		00000C82	R 02
RSB\$K-PARENT	= 00000048						
RSB\$K-VALSEQNUM	= 0000003C						
RSB\$K-WTQBL	= 00000024						
RSB\$K-WTQFL	= 00000020						
RSB\$M-DIRENTRY	= 00000001						
RSB\$M-VALINVL	= 00000002						
RSB\$Q-VALBLK	= 00000028						
RSB\$T-RESNAM	= 00000050						
RSB\$W-BLKASTCNT	= 00000042						
RSB\$W-GROUP	= 0000004C						
RSB\$W-HASHVAL	= 00000044						
RSB\$W-REFCNT	= 00000040						
RSB\$W-ROSEQNM	= 00000046						
RSB\$W-SIZE	= 00000008						
RSB\$W-STATUS	= 0000000E						
RSN\$-CLUSTAN	= 0000000E						
RSN\$-NPDYNMEM	= 00000003						
SCH\$CLREFR	*****	X	02				
SCH\$GETEFC	*****	X	03				
SCH\$GL-CURPCB	*****	X	02				
SCH\$GL-PCBVEC	*****	X	02				
SCH\$POSTEF	*****	X	02				
SCH\$QAST	*****	X	02				
SCH\$REMOVACB	*****	X	02				
SCH\$RWAiT	*****	X	02				
SCH\$SWAPACBS	*****	X	03				
SNDDEQ-GRNT	00000ACE	R	02				
SNDDEQ-WAIT	00000A65	R	02				
SS\$-ABORT	= 0000002C						
SS\$-ACCVIO	= 0000000C						
SS\$-BADPARAM	= 00000014						
SS\$-CANCEL	= 00000830						
SS\$-CANCELGRANT	= 00000E2A						
SS\$-CVTUNGRANT	= 0000213C						
SS\$-EXASTLM	= 00002A04						
SS\$-EXDEPTH	= 00000E1A						
SS\$-EXENQLM	= 00002A44						

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
LOCKMGR	00000D61 (3425.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG
YSEXEPAGED	000002B7 (695.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.06	00:00:01.15
Command processing	107	00:00:00.55	00:00:03.75
Pass 1	493	00:00:20.32	00:00:58.45
Symbol table sort	4	00:00:02.47	00:00:06.45
Pass 2	408	00:00:08.00	00:00:26.62
Symbol table output	1	00:00:00.25	00:00:00.40
Psect synopsis output	0	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1044	00:00:31.69	00:01:36.86

The working set limit was 2100 pages.

125025 bytes (245 pages) of virtual memory were used to buffer the intermediate code.

There were 90 pages of symbol table space allocated to hold 1421 non-local and 220 local symbols.

3775 source lines were read in Pass 1, producing 35 object records in Pass 2.

30 pages of virtual memory were used to define 29 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
\$_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	17
\$_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	9
TOTALS (all libraries)	26

1443 GETs were required to define 26 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LISS:SYSENQDEQ/OBJ=OBJ:SYSENQDEQ MSRC\$:SYSENQDEQ/UPDATE=(ENH\$:SYSENQDEQ)+EXECMLS/LIB

0383 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

SYSRMPSC
LIS

SYSDCLEXH
LIS

SYSDEVALC
LIS

SYSCURTIM
LIS

SYSDBGLSC
LIS

SYSENQDEQ
LIS

SYSDCLCMH
LIS

SYSDERLMB
LIS

SYSDBSSGN
LIS

SYSDELPRC
LIS

0384 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

